KONSEP DAN APLIKASI PEMROGRAMAN MENGGUNAKAN BORLAND C++ BUILDER 6

BAGIAN 1:
APLIKASI KONSOL

M. FACHRURROZI

Daftar Isi

Muqaddimah

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ
اَلْحَمْدُ لِلّٰهِ رَبِّ الْعَالَمِيْنَ. اَللّٰهُمَّ صَلِّ عَلَى سَيِّدِنَا مُحَمَّدٍ
وَعَلَى اٰلِهِ وَصَحْبِهِ وَبَارِكْ وَسَلِّمْ.

وَقَالَ اللهُ تَعَالَى: يَرْفَعِ اللهُ الَّذِيْنَ اٰمَنُوْا مِنْكُمْ وَالَّذِيْنَ أُوْتُوا الْعِلْمَ دَرَجَاتٍ.

“Allah mengangkat derajat orang yang beriman dan orang yang berilmu pengetahuan beberapa derajat”. (Mujaddalah 11)

وَعَنْ أَبِي هُرَيْرَةَ رَضِيَ اللهُ عَنْهُ قَالَ: قَالَ رَسُوْلُ اللهِ صَلَّى اللهُ عَلَيْهِ
وَسَلَّمْ: مَنْ سُئِلَ عِلْمٍ فَكَتَمَهُ أُلْجِمَ يَوْمَ الْقِيَامَةِ بِلِجَامٍ مِنْ نَارٍ.

“Abu Hurairah r.a. berkata: Rasulullah SAW bersabda: Barang siapa yang ditanya suatu ilmu agama lalu menyembunyikannya, maka akan dikendalikan mulutnya pada hari kiamat dengan kendali dari api neraka”. (Abu Dawud, Attirmidzi)

“Tiada akan pernah mampu seseorang dalam mengerjakan sesuatu tanpa pernah mencobanya terlebih dahulu”.

Dari ketiga sumber ilmu inilah penulis ingin berusaha membuat sesuatu yang bermanfaat bagi orang lain, walaupun masih banyak kekurangan yang terdapat di dalam buku ini.

Sebagian besar isi dari buku ini merupakan rangkuman dari sumber-sumber yang telah dibuat penulis lain. Penulis berharap agar buku ini dapat bermanfaat bagi semua kalangan pembaca. Terima kasih untuk semuanya yang telah memberikan banyak kritik dan saran serta dukungan dalampenulisan buku ini. Dunia akan selalu indah karena kejujuran dan kebersamaan.

Palembang,   Juni 2005

Penulis

......untuk si kecil.....

1. Pendahuluan

Seiring dengan perkembangan zaman, sebuah sistem informasi terus menerus menjadi sorotan dan kajian publik dunia. Dalam kurun waktu 20 tahun terakhir, sistem informasi terus dikembangkan guna memenuhi kebutuhan akan sebuah kemudahan dan efektivitas dalam kehidupan. Sering manusia menghadapi kejenuhan terhadap pekerjaannya, membuat mereka menjadi kurang efektif dalam mengatur waktu hidupnya, sehingga banyak kalangan saintis terus berfikir apakah kehidupan akan selalu monoton seperti itu. Akibatnya tercipta sebuah pemikiran untuk membuat sesuatu yang mampu membantu manusia dalam mengerjakan beberapa pekerjaannya. Bahasa pemrograman dikenalkan pada tahun 1967 oleh Martin Richards, yaitu BCPL yang merupakan akar bahasa C sekarang ini. Kemudian berdasar pada bahasa BCPL ini ***Ken Thompson*** yang bekerja di Bell Telephone Laboratories (Bell Labs) mengembangkan bahasa B pada tahun 1970. Saat itu bahasa B telah berhasil diimplementasikan di komputer DEC PDP-7 dengan operating system (OS) UNIX. Pada tahun 1972, peneliti lain di Bell Labs bernama ***Dennis Ritchie*** menyempurnakannya menjadi bahasa C. Pada tahun 1978, ***Dennis Ritchie*** bersama dengan ***Brian Kernigha***n mempublikasikan buku yang kemudian menjadi legenda dalam sejarah perkembangan bahasa C, yang berjudul ***The C Programming Language***. Buku ini diterbitkan oleh Prentice Hall, dan pada saat ini telah diterjemahkan dalam berbagai bahasa di dunia. Boleh dikatakan bahwa buku ini adalah buku yang paling banyak direfer orang dan dijadikan buku panduan tentang pemrograman bahasa C sampai saat ini. Teknik dan gaya penulisan bahasa C yang merefer kepada buku ini kemudian terkenal dengan sebutan ***K&R C*** atau ***Classic C*** atau ***Common C***. Seiring dengan berkembang pesatnya bahasa C, banyak vendor mengembangkan kompiler C menurut versi masing-masing. Hal ini menggerakkan ANSI (***American National Standards Institute***) pada tahun 1983 untuk membuat suatu komite yang kemudian diberi nama ***X3J11***, yang betujuan untuk membuat definisi standar bahasa C yang lebih modern dan komprehensif, dengan memperbaiki ***syntax*** dan ***grammar*** bahasa C. Usaha ini berhasil diselesaikan 5 tahun kemudian, yaitu ditandai dengan lahirnya standard ANSI untuk bahasa C yang kemudian terkenal dengan sebutan ***ANSI C*** pada tahun 1988.

MENGAPA PAKAI BAHASA C++?

Sampai saat ini, bahasa C telah berhasil digunakan untuk mengembangkan berbagai jenis permasalahan pemrograman, dari level operating system (unix, linux, ms dos,

dsb), aplikasi perkantoran (text editor, word processor, spreadsheet, dsb), bahkan sampai pengembangan sistem pakar (***expert system***). Kompiler C juga telah tersedia di semua jenis platform komputer, mulai dari Macintosh, UNIX, PC, Micro PC, sampai super komputer. C bisa disebut bahasa pemrograman tingkat menengah (***middle level programming language***). Arti tingkat (level) disini adalah kemampuan mengakses fungsi-fungsi dan perintah-perintah dasar bahasa mesin/hardware (***machine basic instruction set***). Semakin tinggi tingkat bahasa pemrograman (misalnya: java), semakin mudahlah bahasa pemrograman dipahami manusia, namun membawa pengaruh semakin berkurang kemampuan untuk mengakses langsung instruksi dasar bahasa mesin. Demikian juga sebaliknya dengan bahasa pemrograman tingkat rendah (misalnya: assembler), yang semakin sulit dipahami manusia dan hanya berisi perintah untuk mengakses bahasa mesin. Dalam perspektif mudahnya dipahami manusia, C bisa digolongkan dalam bahasa tingkat tinggi, namun C juga menyediakan kemampuan yang ada pada bahasa tingkat rendah, misalnya operasi bit, operasi byte, pengaksesan memori, dsb.

Beberapa alasan mengapa memakai bahasa C adalah terangkum dibawah.

1. ***C adalah bahasa pemrograman yang paling populer saat ini***
   Dengan banyaknya programmer bahasa C, membawa pengaruh semakin mudahnya kita menemukan pemecahan masalah yang kita dapatkan ketika menulis program dalam bahasa C. Pengaruh positif lain adalah semakin banyaknya kompiler yang dikembangkan untuk berbagai platform (berpengaruh ke portabilitas).
2. ***C adalah bahasa pemrograman yang memiliki portabilitas tinggi***
   Program C yang kita tulis untuk satu jenis platform, bisa kita kompile dan jalankan di platform lain dengan tanpa ataupun hanya sedikit perubahan. Ini bisa diwujudkan dengan adanya standarisasi ANSI untuk C.
3. ***C adalah bahasa pemrograman dengan kata kunci (keyword) sedikit***
   Kata kunci disini adalah merupakan fungsi ataupun kata dasar yang disediakan oleh kompiler suatu bahasa pemrograman. Hal ini membawa pengaruh semakin mudahnya kita menulis program dengan C. Pengaruh lain dari sedikitnya kata kunci ini adalah proses eksekusi program C yang sangat cepat. C hanya menyediakan 32 kata kunci seperti terangkum dibawah:

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigned	void	volatile	while			

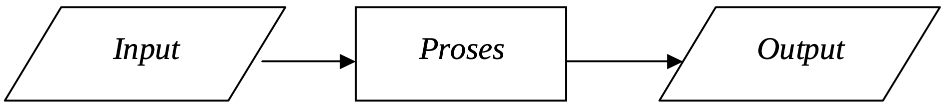
4. ***C adalah bahasa pemrograman yang fleksibel***
   Dengan menguasai bahasa C, kita bisa menulis dan mengembangkan berbagai jenis program mulai dari operating system, word processor, graphic processor, spreadsheets, ataupun kompiler untuk suatu bahasa pemrograman.
5. ***C adalah bahasa pemrograman yang bersifat moduler***
   Program C ditulis dalam ***routine*** yang biasa dipanggil dengan fungsi. Fungsi-fungsi yang telah kita buat, bisa kita gunakan kembali (***reuse***) dalam program ataupun aplikasi lain.

2. Konsep Dasar Pemrograman

PENDAHULUAN KONSEP PEMROGRAMAN

Setiap sebelum melakukan pekerjaan, sangat diperlukan sekali sebuah pengenalan terhadap konsep pekerjaan tersebut guna memahami mencapai target pekerjaan yang dibuat nanti, akan lebih mudah dikerjakan apabila permasalahan dan alur pekerjaan jelas dan dimengerti. Misalkan seseorang ingin mendapatkan hasil ujian yang baik, yang harus ia lakukan adalah mencari sebuah rumusan atau cara agar ia mencapai targetnya. Pertama yang harus dicari agar hasil ujian baik adalah, materi-materi mengenai ujian tersebut harus dipahami dan dikuasai, kemudian bagaimana agar materi-materi ujian tersebut bisa dikuasai, yaitu dengan cara membaca dan belajar dari sumber-sumber buku mengenai materi tersebut. Dari contoh ini bisa terlihat, bahwa output yang diinginkan adalah mendapatkan hasil nilai yang baik, proses yang harus dilewati adalah pada saat ujian tersebut, lalu inputnya adalah belajar dan membaca dari sumber buku-buku agar materi ujian dapat dipahami dan dikuasai. Contoh kedua, misalkan seseorang akan pergi ke kota B dari kota A, artinya target (output) yang ia harus capai adalah mencapai kota B. Kemudian bagaimana ia bisa mencapai kota B, yaitu dengan menggunakan bis dari kota A ke kota B, dan ini merupakan proses bagaimana ia mencapai kota B. Sekarang apa yang harus ia gunakan agar bisa menggunakan bis untuk mencapai kota B?. Bis bisa digunakan apabila ia memiliki dan memberikan uang ke sopirnya, dan uang ini sebagai inputnya.

Seperti yang digambarkan di atas, konsep kerja dan logika pemrograman harus dipahami terlebih dahulu. Sebuah pemrograman tidak terlepas dari konsep kerja sebuah komputer, terdapat logika dasar input, proses dan output, artinya ada data-data yang harus diinput, baik itu secara langsung maupun tidak langsung, selanjutnya akan diproses, lalu akan dioutputkan ke layar maupun ke media lainnya.



Misalkan ada sebuah kasus yaitu mencari luas segitiga sama kaki.

Langkah 1:

Tentukan target akhir (output) dari kasus tersebut, yaitu mencari luas segitiga sama kaki. Misalkan luas segitiga disimbolkan dengan L.

Langkah 2:

Pahami bagaimana perumusan (proses) untuk mendapatkan nilai luas segitiga sama kaki tersebut, yaitu setengah daru luas alasnya dikalikan dengan tinggi segitiga sama kaki tersebut.

Langkah 3:

Tentukan data-data (input) apa saja yang harus diambil guna melakukan perumusannya, yaitu data luas alas dan tinggi segitiga sama kaki. Misalkan luas alas disimbolkan dengan A dan tingginya disimbolkan dengan t.

Untuk pemrogramannya ketiga langkah di atas harus dibaca dari bawah, yaitu:

1. Tentukan data-data (input) apa saja yang harus diambil guna melakukan perumusannya, yaitu data luas alas dan tinggi segitiga sama kaki. Misalkan luas alas disimbolkan dengan A dan tingginya disimbolkan dengan t.

2. Pahami bagaimana perumusan (proses) untuk mendapatkan nilai luas segitiga sama kaki tersebut, yaitu setengah daru luas alasnya dikalikan dengan tinggi segitiga sama kaki tersebut.

$$L = 0{,}5 \cdot A \cdot t$$

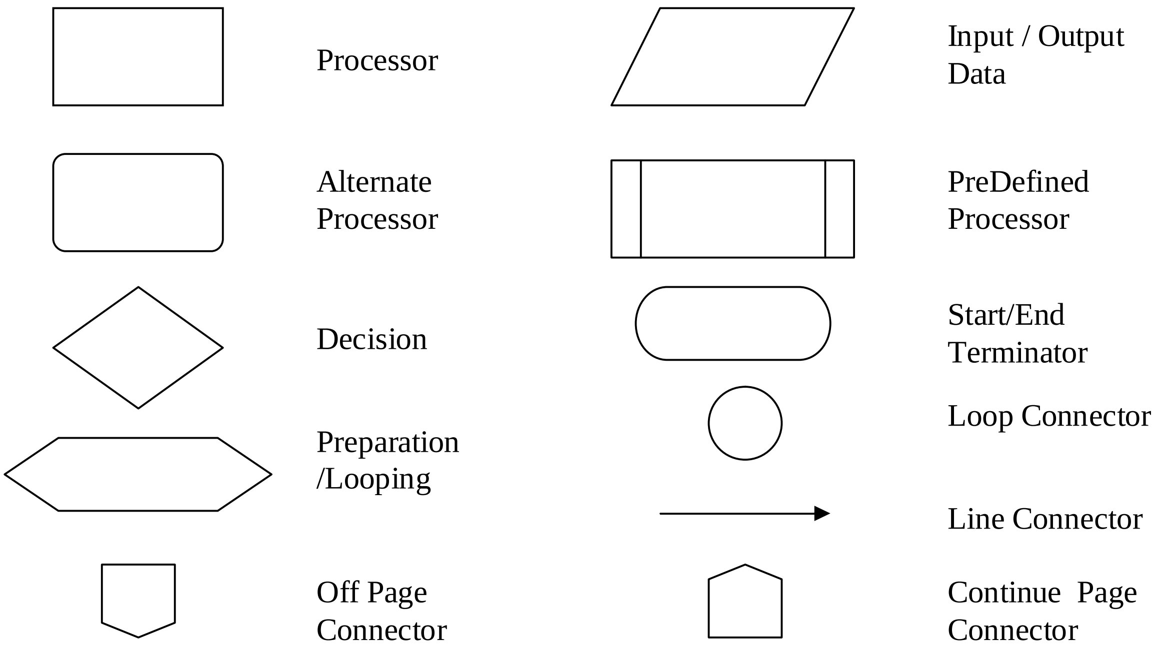
3. Tentukan target akhir (output) dari kasus tersebut, yaitu mencari luas segitiga sama kaki. Misalkan luas segitiga disimbolkan dengan L.

Dari contoh-contoh di atas, dapat diambil kesimpulan bahwa konsep pemrograman tidak terlepas dari data input, proses dan output. Konsep inilah yang menjadi dasar pembuatan sebuah pemrograman.

KONSEP PEMROGRAMAN MENGGUNAKAN DIAGRAM ALIR (FLOWCHART)

Konsep pemrograman dapat digambarkan juga dengan menggunakan diagram alir (***flowchart***). Diagram alir adalah simbol-simbol yang digunakan untuk menggambarkan

sebuah pernyataan logika pemrograman serta aliran logika yang ditunjukkan dengan arah panah. Berikut merupakan beberapa contoh simbol yang disepakati oleh dunia pemrograman:



Untuk memahami lebih dalam mengenai diagram alir ini, akan diambil beberapa sebuah kasus sederhana.

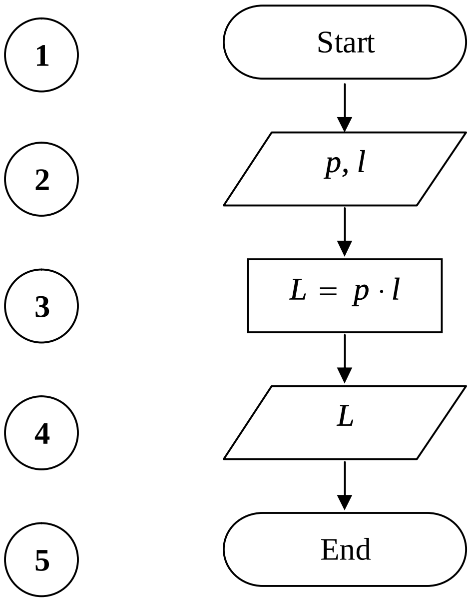
Kasus 1 :
Buatlah sebuah rancangan program dengan menggunakan diagram alir, mencari luas persegi panjang.

Solusi 1:
Perumusan untuk mencari luas persegi panjang adalah

$$L = p \cdot l$$

dimana, ***L*** adalah Luas persegi panjang, ***p*** adalah panjang persegi dan ***l*** adalah lebar persegi.

Flowchart 1:



Keterangan 1:

1. Simbol pertama menunjukkan dimulainya sebuah program.
2. Simbol kedua menunjukkan bahwa input data dari p dan l.
3. Data dari ***p*** dan ***l*** akan diproses pada simbol ketiga dengan menggunakan perumusan $L = p \cdot l$.
4. Simbol keempat menunjukkan hasil output dari proses dari simbol ketiga.
5. Simbol kelima atau terakhir menujukkan berakhirnya program dengan tanda ***End.***

Kasus 2:
Mencari akar-akar persamaan kuadrat $f(x) = ax^2 + bx + c = 0$

Solusi 2:
Telaah masalah terlebih dahulu, yaitu perumusan nilai akar-akar persamaan kuadrat tersebut:

Misalkan

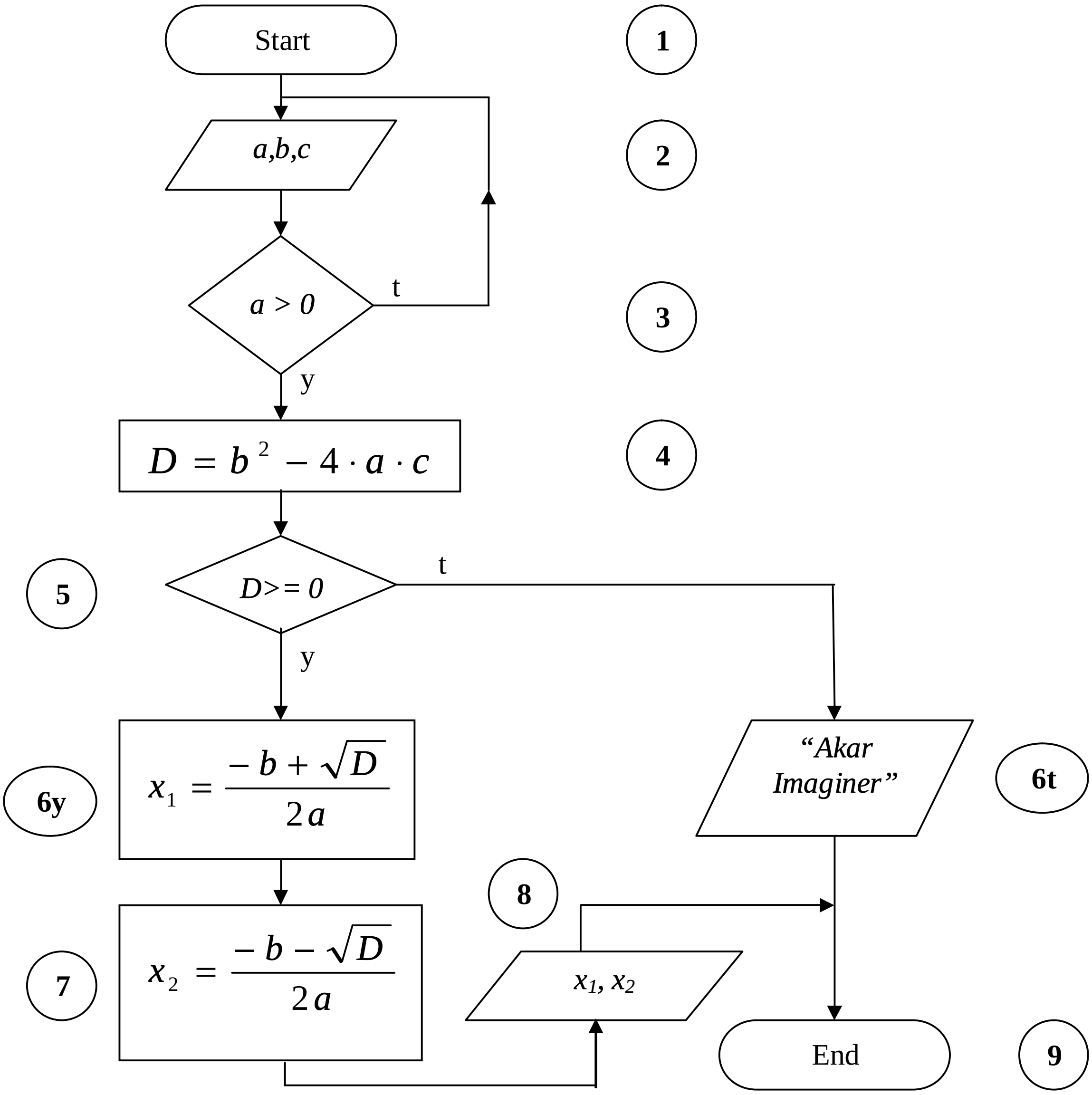
$$D = b^2 - 4ac$$, selanjutnya

jika untuk nilai D >= 0, solusinya adalah

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ dan } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$, untuk nilai $\boldsymbol{a > 0}$.

Selain nilai D>=0 adalah D<0. Karena hasil akar dari D<0 bernilai imaginer, maka solusi akar persamaan tersebut tidak ada.

Flowchart 2:



Keterangan 2:

1. Simbol 1 menunjukkan dimulainya sebuah program.
2. Simbol 2 menunjukkan input data dari ***a***, ***b*** dan ***c***.
3. Simbol 3 menunjukkan suatu keputusan yang akan dipilih, yaitu jika nilai a memenuhi kriteria a > 0, maka akan melanjutkan ke proses selanjutnya. Jika tidak, maka akan kembali ke simbol 2
4. Data dari ***a***, ***b*** dan ***c*** akan diproses pada simbol keempat dengan menggunakan perumusan $D = b^2 - 4 \cdot a \cdot c$.
5. Simbol 5 menunjukkan suatu keputusan yang akan dipilih, jika hasil output dari proses dari simbol 4, yaitu ***D >=0*** , maka akan melanjutkan ke proses di simbol 6y dan 7, jika tidak memenuhi, maka akan melanjutkan ke simbol 6t.
6. Simbol 6y dan 7 adalah proses perhitungan x_1 dan x_2.
7. Simbol 8 adalah nilai output dari proses 6y dan 7
8. Simbol 6t menunjukkan output yang berupa text ***"Akar Imaginer".***
9. Simbol 9 atau terakhir menujukkan berakhirnya program dengan tanda ***End.***

Kasus 3:

Menghitung rata-rata dari n buah bilangan.

Solusi 3:

Telaah masalah terlebih dahulu, yaitu perumusan mencari rata-rata dari n buah bilangan.

$$rata = \frac{jumlah\,data}{banyak\,data}, \text{ atau}$$

$$rata = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}, \text{ atau}$$

$$rata = \frac{\sum_{1}^{n} x_i}{n}$$

Sebelum membuat diagram alirnya, perhatikan dengan teliti perumusan di atas. Terlihat adanya variable x dengan indeks yang berurut (membentuk sebuah deret) yaitu 1, 2, 3, …, n. Artinya setiap terjadi proses yang membentuk sebuah deret, baik itu deret aritmatika, maupun deret geometri, maka proses tersebut dapat dilakukan dengan menggunakan teknik perulangan (looping). Tentukan nilai awal dari perulangan tersebut, dalam hal ini nilai awalnya adalah 1. Kemudian tentukan step, atau selisih, atau

beda, atau rasio dari deret tersebut. Karena deret di atas merupakan deret aritmatika, maka step atau selisihnya adalah 1. Deret indeks di atas berakhir pada saat nilai indeks sama dengan n.

Selanjutnya dari perumusan di atas, bahwa diperlukan nilai jumlah dari semua data. Di dalam pemrograman, untuk menghitung jumlah dari n buah data, maka harus didefinisikan dulu sebuah variabel jumlah dengan nilai awal adalah 0. Kemudian proses penjumlahan akan dilakukan di dalam perulangan dengan perumusan:

$$jumlah = jumlah + x[i]$$

Perhatikan tabel berikut:

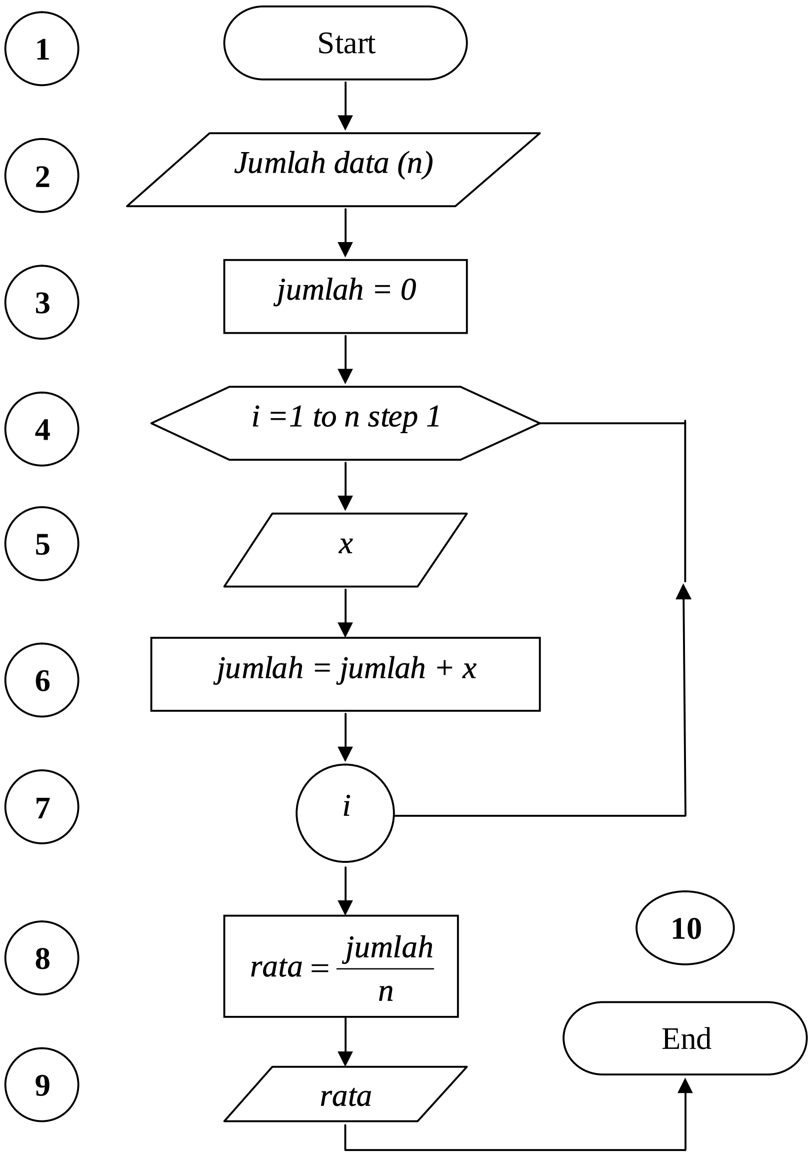
Indeks (i)	X	Jumlah lama	Jumlah baru (jumlah lama + x)
1	1	0	1
2	2	1	3
3	3	2	5
4	4	3	7
5	5	7	12

Tabel 1. Tabel logika perulangan

terlihat posisi ***jumlah baru*** akan menjadi ***jumlah lama*** pada perulangan berikutnya, sehingga untuk nilai n = 5, jumlah akhir adalah 12.

Setelah mendapatkan nilai jumlah, maka akan dapat dihitung nilai rata-rata dari n buah data tersebut.

Flowchart 3:



Keterangan 3:

1. Simbol 1 menunjukkan dimulainya sebuah program.
2. Simbol 2 menunjukkan input jumlah data dengan variabel ***n***.
3. Simbol 3 menunjukkan nilai jumlah awal adalah 0.
4. Simbol 4 menunjukkan sebuah perulangan dengan nilai awal 1, nilai akhir ***n*** dan step 1, digunakan indeks ***i*** sebagai variabel perulangan dan simbol 7 sebagai batas proses yang berulang.

5. Simbol 5 merupakan input data nilai dari variabel x yang akan dijumlahkan dengan jumlah, ditunjukkan pada simbol 6.
6. Simbol 8 menunjukkan proses perhitungan nilai ***rata*** setelah didapat hasil akhir dari perhitungan ***jumlah***, kemudian dioutputkan pada simbol 9.
7. Simbol 10 atau terakhir menujukkan berakhirnya program dengan tanda ***End.***

Kasus 4:

Buat sebuah logika pemrograman menggunakan flowchart untuk menampilkan deret bilangan ganjil 1 – 100 yang habis dibagi dengan 3 atau 5.

Solusi 4:

Telaah masalah terlebih dahulu, yaitu perumusan untuk deret bilangan ganjil yang habis dibagi dengan 3 atau 5.

Output yang akan dihasilkan : ***3 5 9 15 21 25 27 ... dst***

Deret bilangan ganjil: ***1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 ... 2n-1***

Bilangan yang tidak tampil : ***1 7 11 13 17 19 23 29 ... dst***

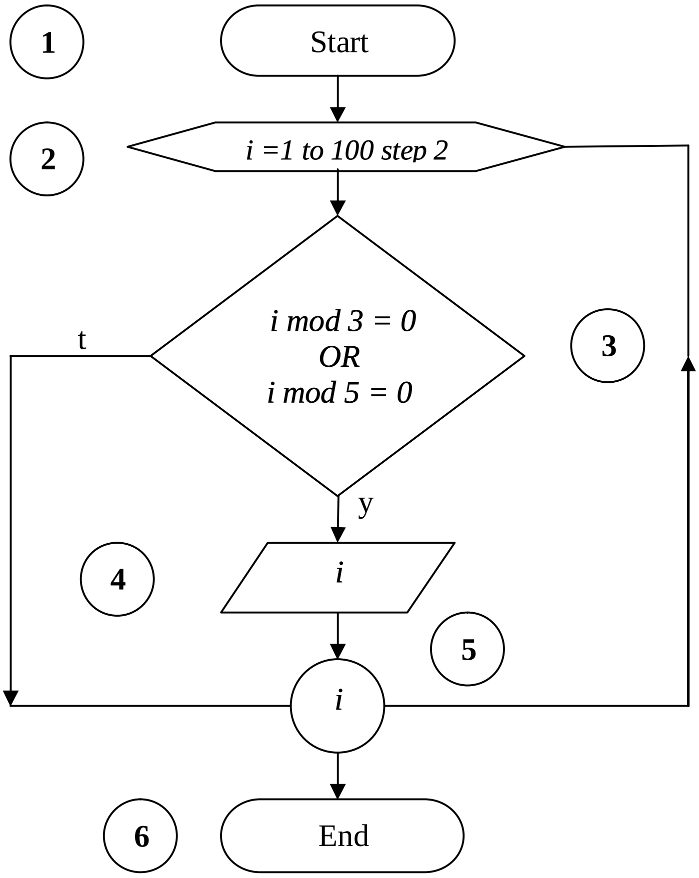
Sebagaimana yang telah dijelaskan sebelumnya, setiap output yang menghasilkan sebuah deret, maka akan terjadi proses berulang pada programnya. Dari deret yang ditentukan, pertama lakukan proses untuk output deret bilangan ganjilnya, karena deret tersebut merupakan deret utama dari kasus di atas dan juga deret yang memiliki indeks yang berurut, dengan ***<u>nilai awalnya 1, batas akhirnya 100 dengan step 2</u>***. Setelah dibuat untuk proses deret bilangan ganjilnya saja, kemudian kondisikan bilangan yang tampil, yaitu bilangan yang hanya habis dibagi dengan 3 atau 5. Sebelumnya harus dipahami dulu makna logika pemilihan dengan menggunakan operator logika. Lihat tabel di bawah berikut:

P	Q	P AND Q	P OR Q
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	TRUE

Tabel 2. Tabel Kebenaran

Dari kasus di atas, tedapat kondisi yang harus dipenuhi, yaitu kondisi hanya dapat dibagi dengan 3 atau 5, artinya semua bilangan ganjil dari 1 – 100 yang dapat dibagi dengan 3 ***atau*** 5 akan tampil pada output.

Flowchart 3:



Keterangan 4:

1. Simbol 1 menunjukkan dimulainya sebuah program.
2. Simbol 2 menunjukkan sebuah perulangan dengan nilai awal ***1***, nilai akhir ***100*** dengan step ***2***, digunakan indeks ***i*** sebagai variabel perulangan dan simbol 5 sebagai batas proses yang berulang.
3. Simbol 3 menunjukkan proses pemilihan kondisi, jika memenuhi kondisi di simbol 3, maka akan menampilkan nilai dari ***i*** di simbol 4, jika tidak maka akan mengulang proses dengan indeks ***i*** selanjutnya.
4. Simbol 6 atau terakhir menujukkan berakhirnya program dengan tanda ***End.***

Dari keempat contoh kasus di atas, jelaslah bahwa diagram alir (flowchart) dapat menunjukkan logika proses berjalannya suatu pemrograman. Konsep runtunan, pemilihan dan perulangan perlu dipelajari dan dipahami terlebih dahulu sebelum melangkah ke proses pemahaman pemrograman selanjutnya.

3. Instalasi dan Pengenalan Borland C++ Builder 6

Setelah mempelajari dan memahami konsep pemrograman, baik secara pemahaman lewat cerita maupun dengan flowchart, maka akan dibahas mengenai implementasi konsep pemrograman ke dalam bahasa pemrograman menggunakan bahasa ***C++***, dalam hal ini akan menggunakan software Borland C++ Builder 6.

INSTALASI BORLAND C++ BUILDER 6

Sebelum memulai instalasi software ini, siapkan cd#1 dan cd#2 atau ***master source*** Borland C++ Builder 6.

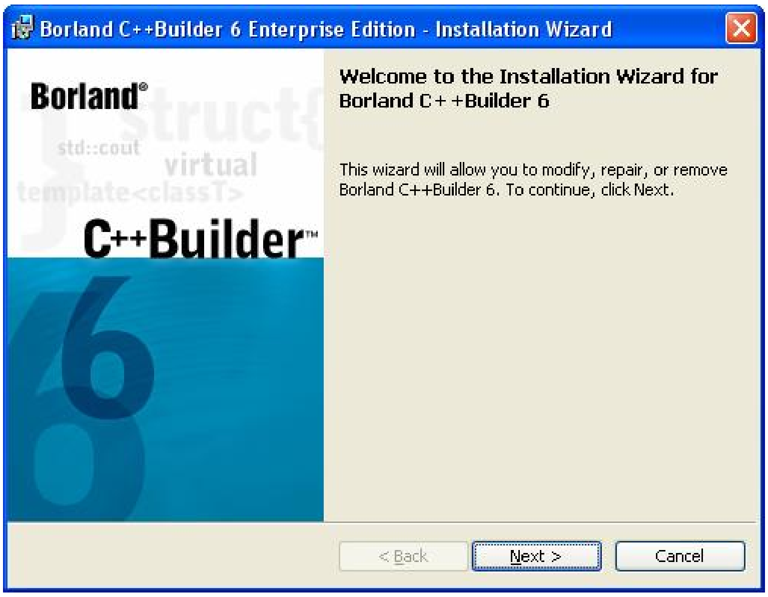
Langkah-langkah:

1. Masukkan cd#1, lalu secara otomatis cd akan menjalankan program autorun. Atau klik install.exe di master sourcenya. Akan muncul tampilan berikut:



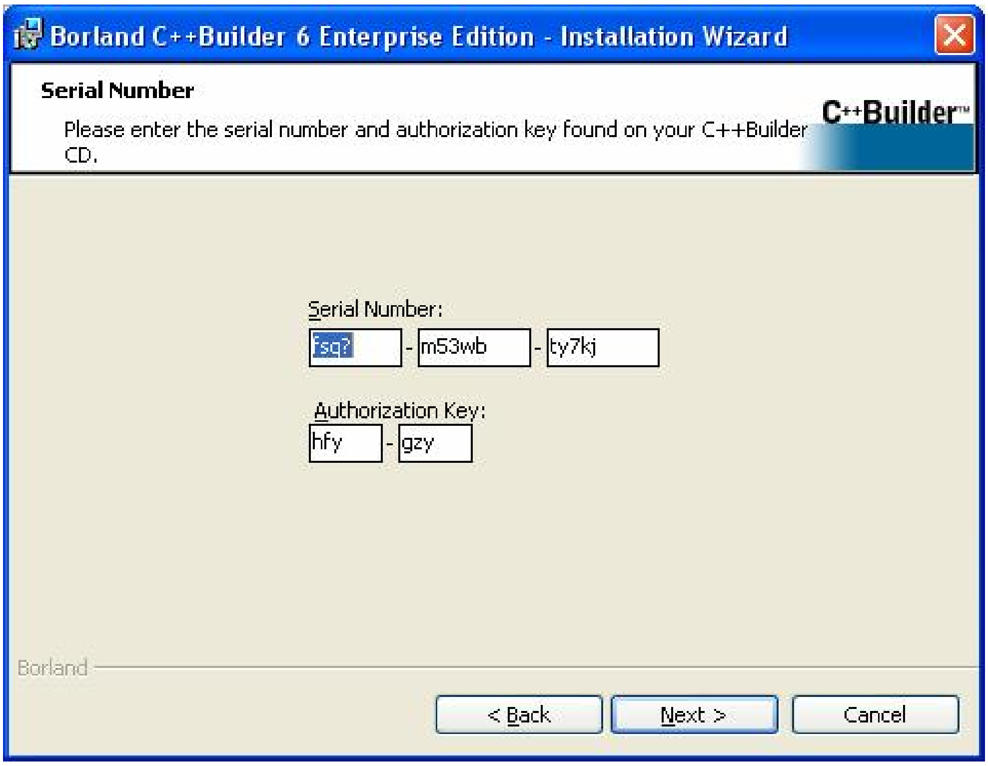
Gambar 3.1 Launcher

2. Pilih menu yang pertama, yaitu C++ Builder 6. akan muncul tampilan berikut ini:



Gambar 3.2 Instalation Wizard

Klik Next, akan muncul tampilan berikut ini:



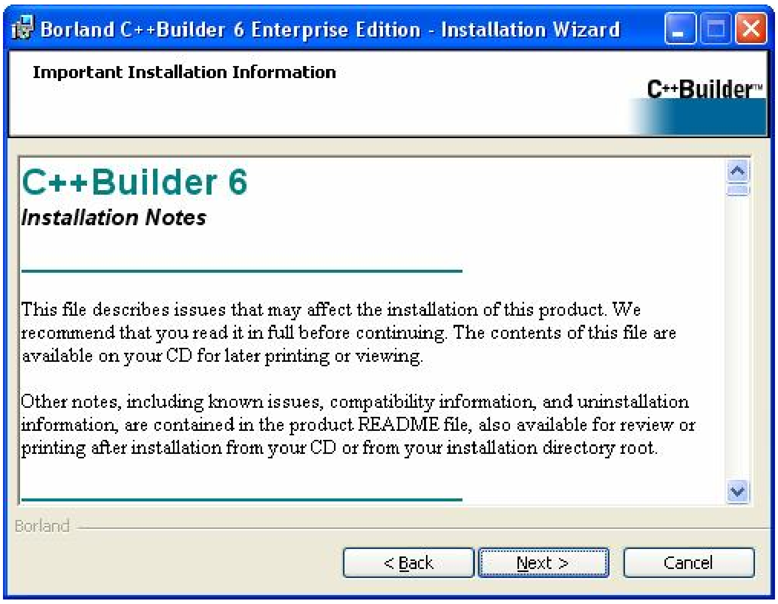
Gambar 3.3 Serial Number

3. Isi Serial Number dan Authorization key. Lalu klik Next, aka muncul tampilan berikut ini:



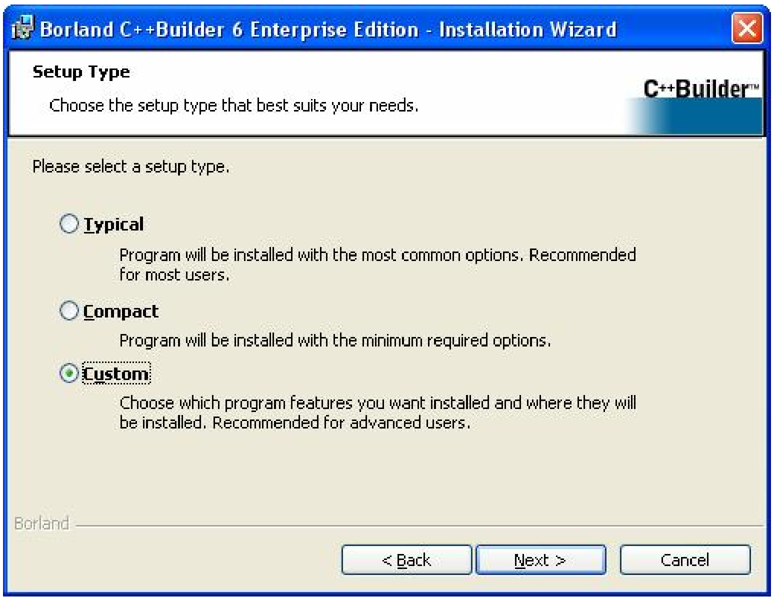
Gambar 3.4 License Agreement

Setelah membaca License Agreement, pilih opsi pertama "I accep the terms in the license agreement", lalu klik Next, akan mucul tampilan berikut ini:



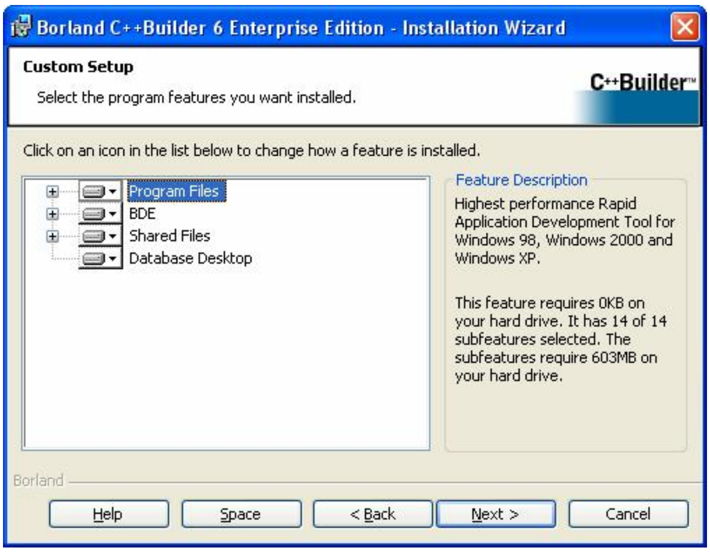
Gambar 3.5 Information

Klik Next, lalu muncul tampilan berikut ini:



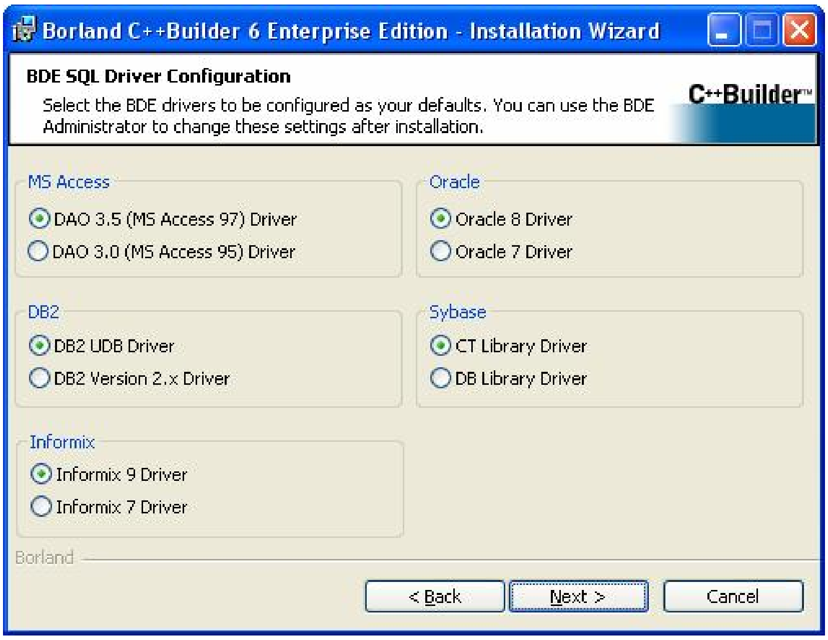
Gambar 3.6 Setup Type

Pada tampilan ini akan diminta untuk jenis instalasi yang diinginkan, tapi dianjurkan untuk memilih ***Custom Instalation,*** karen semua fitur akan terinstal, termasuk file Help nya yang akan membantu dalam pencarian bantuan yang berhububungan dengan Borland C++ Buider 6. Klik Next untuk lanjut ke tampilan berikut:



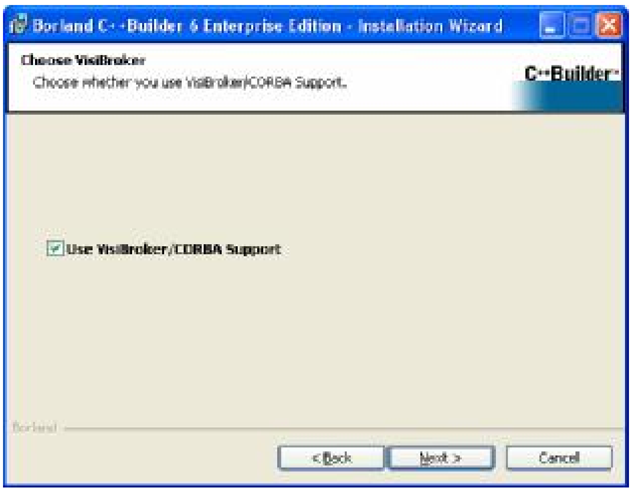
Gambar 3.7 Custom Setup

4. Biarkan semua komponen terpilih (keadaan warna putih semua). Klik Next untuk lanjut ke tampilan berikut:



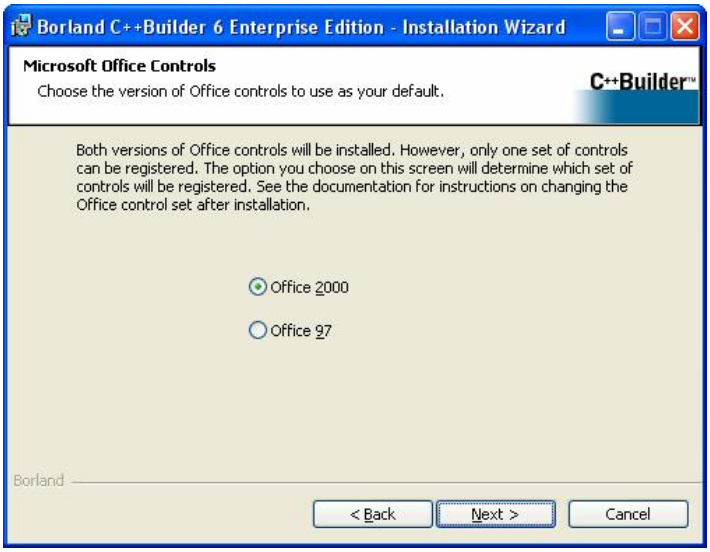
Gambar 3.8 BDE SQL Driver Configuration

5. Pilih sesuai tipe driver yang telah terpasang di komputer, namun jika tidak tahu, klik pilih saja yang paling baru atau jangan diubah sama sekali pilihannya (default), kemudian klik Next untuk tampilan berikut ini:



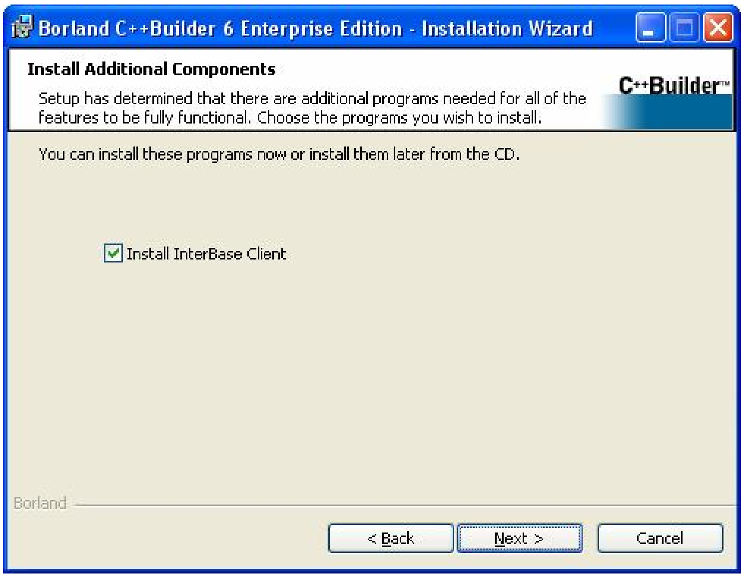
Gambar 3.9 Choose Visibroker

Klik pilihan ini jika ingin menggunakan VisiBroker/CORBA Support. Klik Next untuk lanjut ke tampilan berikut:



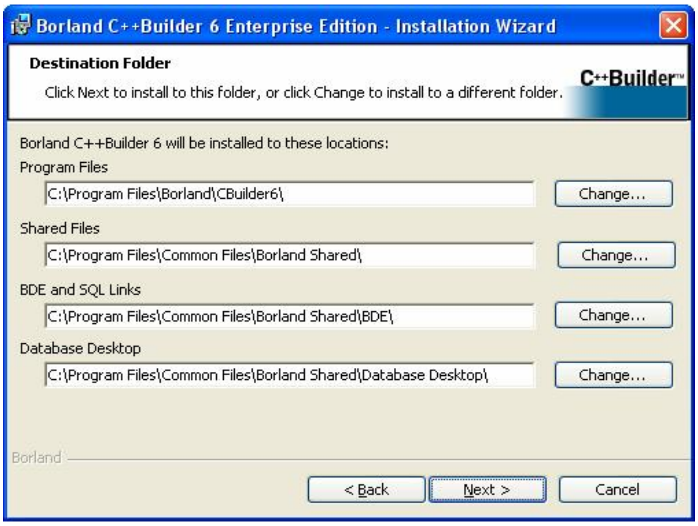
Gambar 3.10 Microsoft Office Control

6. Klik Office 2000 jika di dalam komputer telah terinstal MS Office 2000 ke atas, lalu klik Next untuk lanjut ke tampilan berikut:



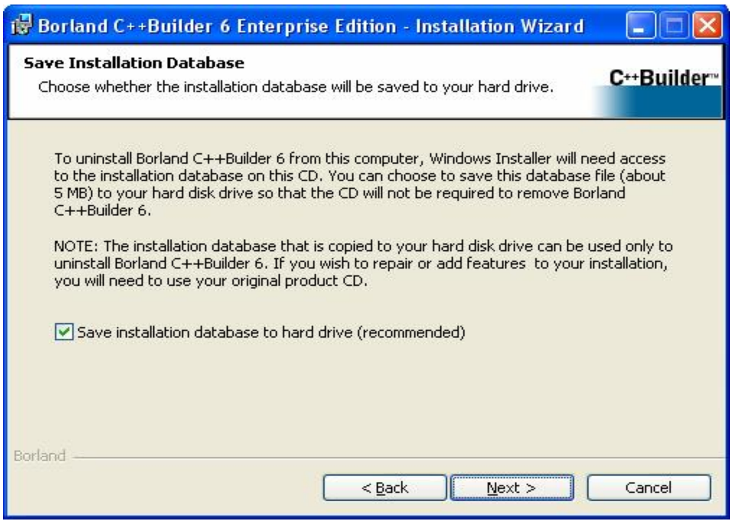
Gambar 3.11 Install Additional Components

7. Pilih fitur ini jika ingin menggunakan jenis database InterBase Client. InterBase merupakan salah satu jenis database yang mendukung aplikasi database client-server atau jaringan. Setelah itu akan diminta persetujuan kesepakatan untuk menginstalnya. Klik Next dan Next lagi untuk lanjut ke tampilan berikut:



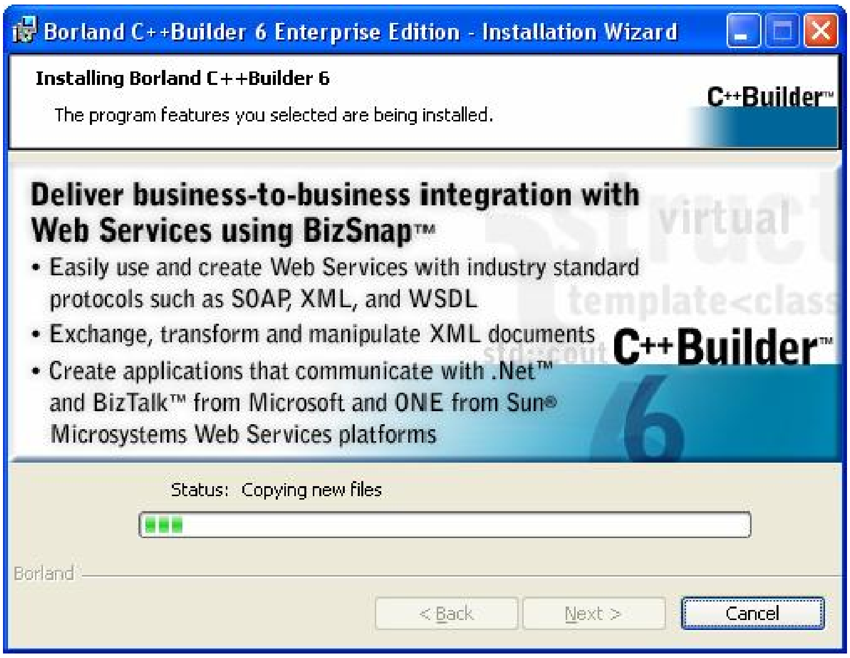
Gambar 3.12 Destination Folder

8. Biarkan tempat-tempat tujuan instalasi seperti semula (default), karena akan lebih mudah untuk pembelajaran berikutnya dan buku-buku yang dibuat akan mengarah pada tempat yang semula. Klik Next untuk lanjut ke tampilan berikut:

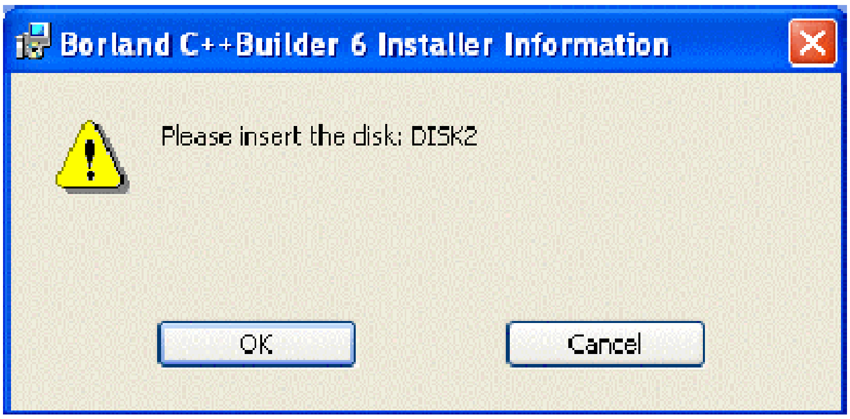


Gambar 3.13 Save Instalation Database

9. Biarkan pilihan default, karena pada saat software ini akan di uninstal, akan diminta CD master sourcenya, namun jika space harddisk tidak mencukupi, maka jangan pilih fitur ini. Klik Next dan Next untuk memulai instalasi.

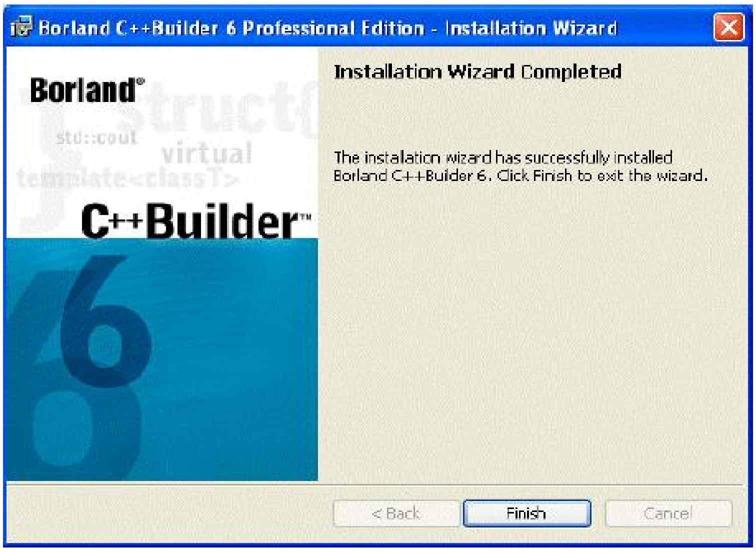


Gambar 3.14 Installing Progress



Gambar 3.15 Insert Disc 2

10. Masukkan CD#2 dan klik OK untuk lanjut proses instalasi. Ikuti saja setiap proses instalasi lainnya (Java, VisiBroker, Interbase) setelah proses instalasi Borland C++ Builder 6 selesai.



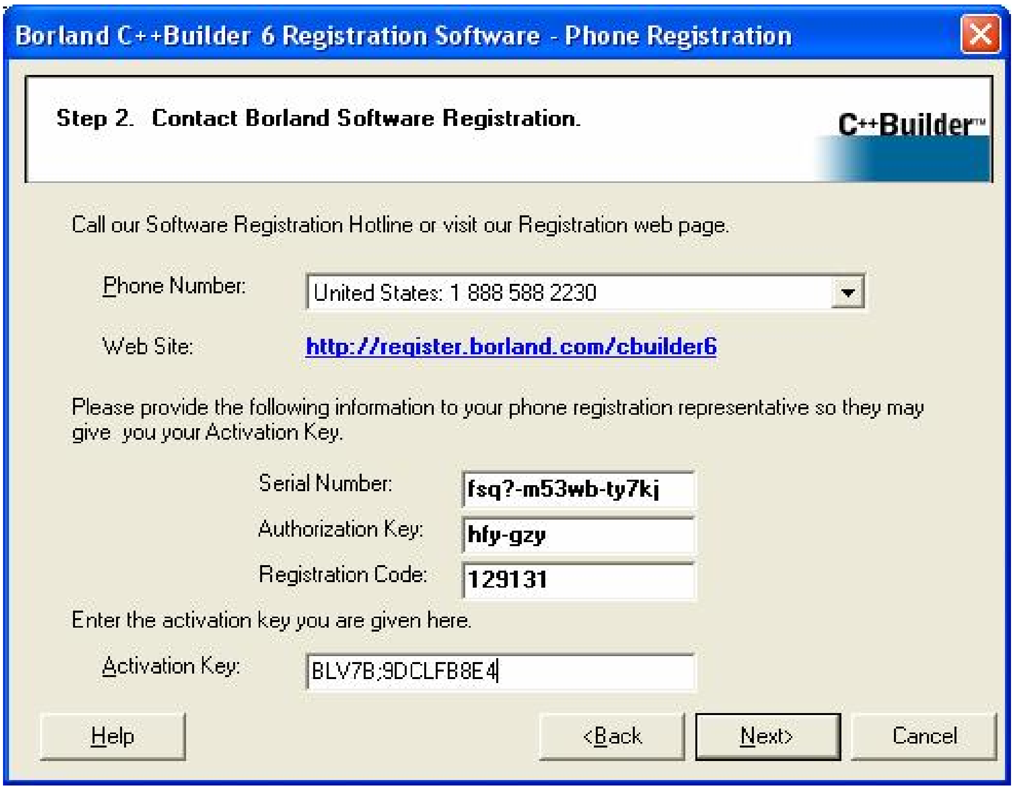
Gambar 3.16 Instalation Completed

11. Setelah muncul tampilan ini, maka instalasi selesai, reboot komputer.
12. Sebelum memulai membuka software Borland C++ Builder 6, akan diminta untuk mengisi kode aktivasi. Isi sesuai nomor yang diperoleh, seperti pada tampilan berikut:



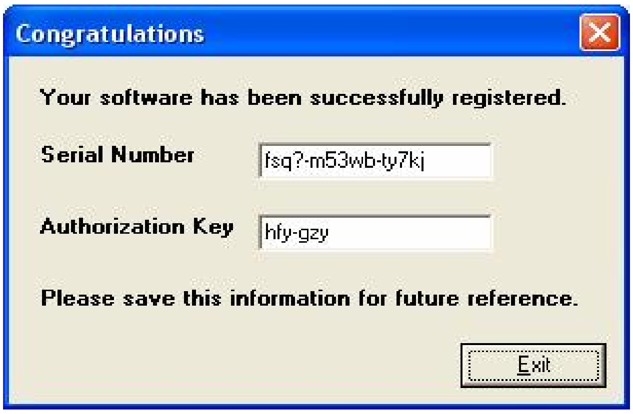
Gambar 3.17 Registration

13. Pilih yang kedua untuk memasukkan kode aktivasi dari telepon, dan masukkan kode tersebut seperti pada tampilan berikut:



Gambar 3.18 Activation Key

Klik Next untuk melihat apakah kode yang dimasukkan benar atau tidak.

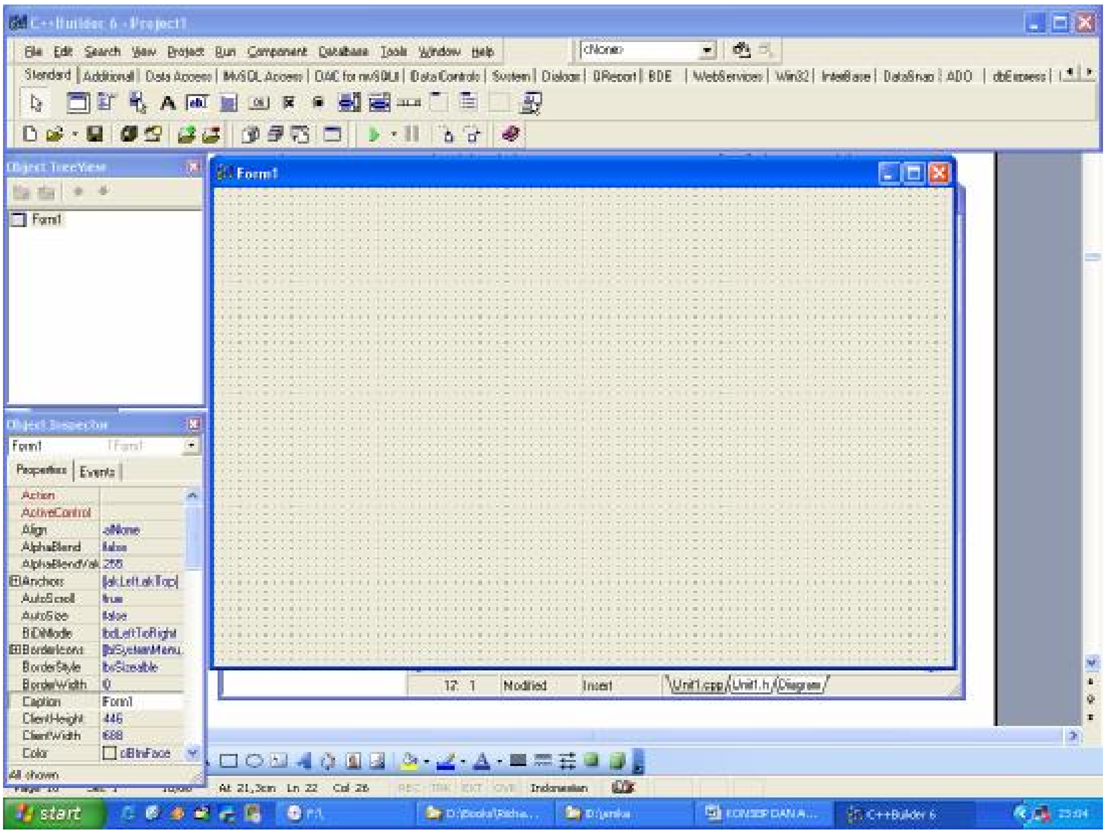


Gambar 3.19 Congratulations

14. Klik Exit setelah registrasi sukses, saatnya memulai menggunakan software Borland C++ Builder 6.

Borland C++ Builder 6 merupakan software yang mendukung sebagain besar kebutuhan dari seorang programmer, baik dari segi fasilitas (features) maupun daya dukungnya di setiap lini kebutuhan. Programmer dapat membuat beberapa jenis aplikasi, diantaranya adalah aplikasi konsol (console application) yang berbasi dos dan aplikasi visual (visual appplication) yang berbasi window. Di tahap awal ini hanya akan dikenalkan beberapa fitur yang mendukung aplikasi konsolnya saja.

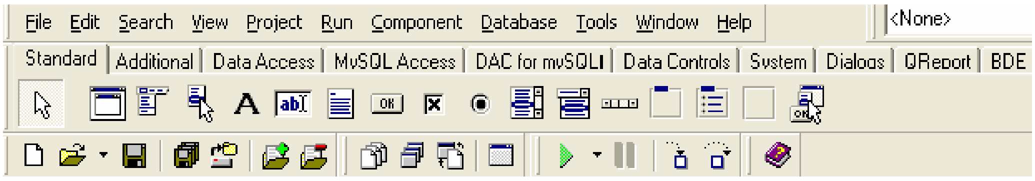
Pada bagian pertama setelah dibuka software Borladn C++ Builder 6, akan muncul tampilan sebagai berikut:



Gambar 3.20 Tampilan awal

Pada tampilan awal ini, ada beberapa bagian yang akan dikenalkan, yaitu:

1. Menu dan komponen



Gambar 3.21 Menu dan Component

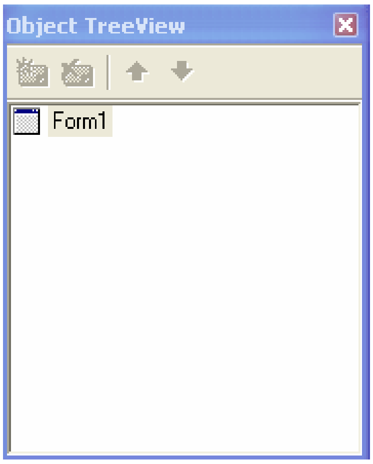
Pada barisan menu (terdiri dari File, Edit, Search,...) berfungsi sebagai tempat mengatur dan menjalankan berbagai perintah dan fitur yang berhubungan dengan Borland C++ Builder 6, termasuk untuk perintah Compile dan Run Program

Pada barisan tab component pallete (terdiri dari Standard, Additional, Data Acces,...) berfungsi saat menggunakan aplikasi visual, yaitu untuk penambahan komponen-komponen input, proses dan output yang berbentuk visual.

Pada barisan ketiga, merupakan isi dari masing-masing tab komponen

Pada barisan keempat, merupakan toolbar-toolbar yang juga berfungsi sebagai langkah cepat dalam menggunakan fitur di Borland C+ Builder 6, seperti toolbar Run dan toolbar new .

2. Object Treeview

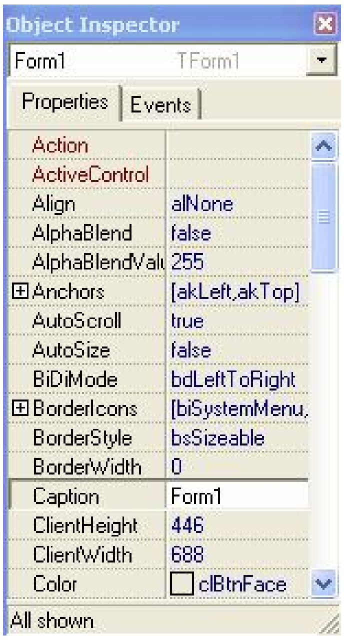


Pada Bagian ini menunjukkan beberapa komponen yang terdapat di dalam Form. Bagian ini akan menyesuaikan jika mengubah ke Form yang lain

Gambar 3.22 Object Treeview

3. Object Inspector

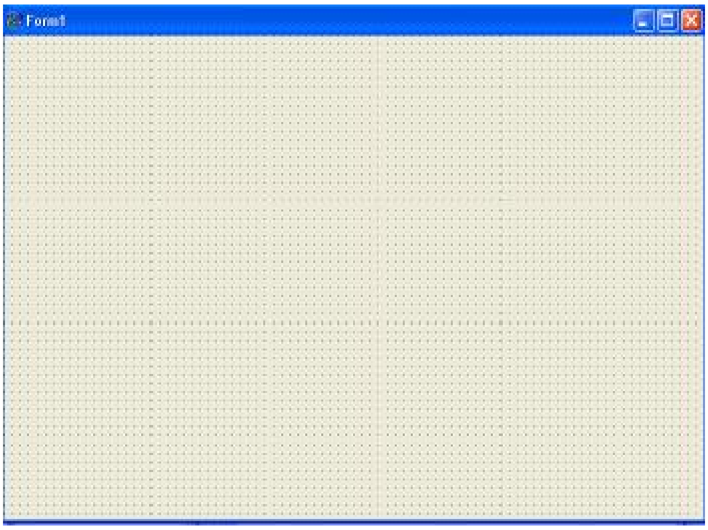
Pada bagian ini menunjukkan properti-properti dan event-event proses yang terdapat di komponen tertentu.



Gambar 3.23 Object Inspector

4. Form

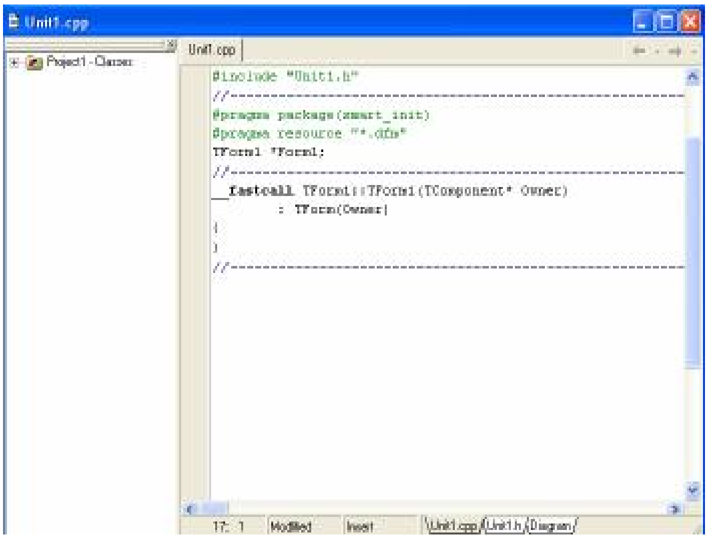
Pada bagian ini adalah tempat meletakkan komponen-komponet yang juga merupakan tempat mengatur logika berpikir secara visual.



Gambar 3.24 Form

5. Source Code

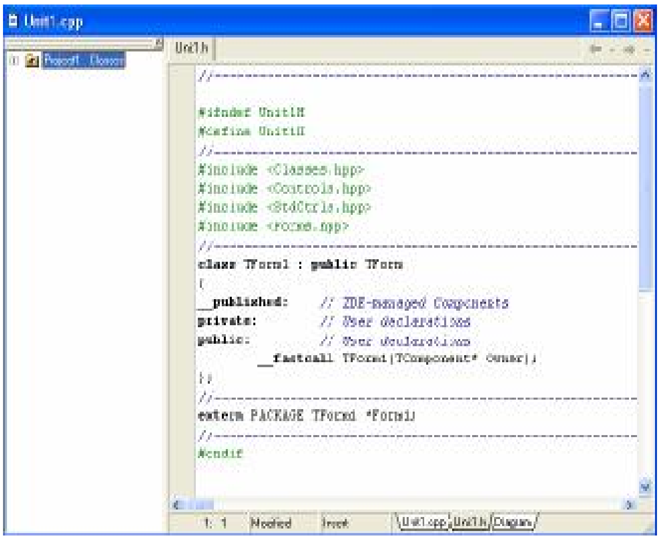
Bagian ini tempat mengisikan sintak-sintak yang akan menjalan kan aplikasi yang akan dibuat, yang berupa file *.cpp.



Gambar 3.6 Source Code

6. Header Unit

Pada bagian ini akan terbentuk secara otomatis pada saat memilih aplikasi yang berbentuk visual. Bagian ini berfungsi sebagai tempat pembuatan class-class yang akan digunakan di bagian Source Code.



Gambar 4.25 Header Unit

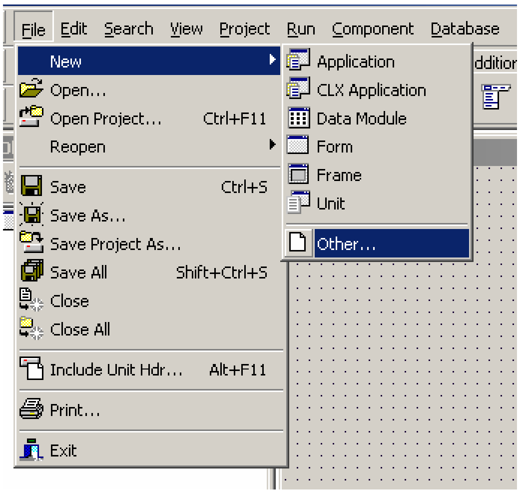
Untuk lebih jelas dan lebih dekat dengan Borland C++ Builder 6, lanjutkan ke bab berikutnya, karena dengan sering latihan dan menggunakannya, maka akan lebih kenal dan dekat dengan lingkungan Borland C++ Builder 6 ini.

4. Pemrograman Dengan Bahasa C++ Console Application

MEMBUAT APLIKASI BARU

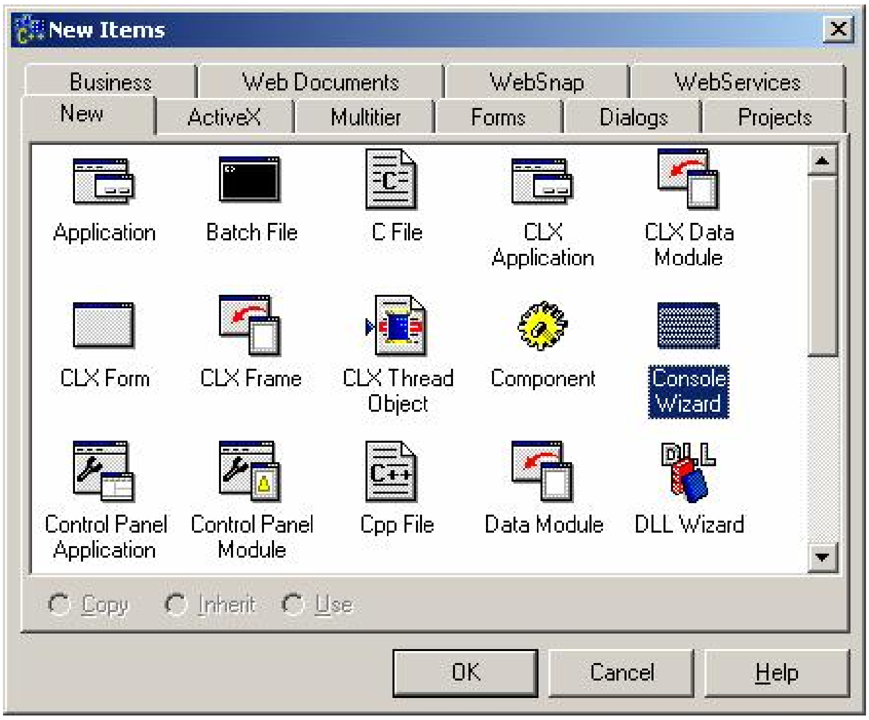
Untuk membuat aplikasi baru, langkah – langkah yang harus dilakukan adalah:

1. Klik menu file, lalu pilih Close all.
2. Klik menu file, lalu pilih new, kemudian pilih Other...



Gambar 4.2 Menu awal

lalu akan muncul gambar di bawah ini:



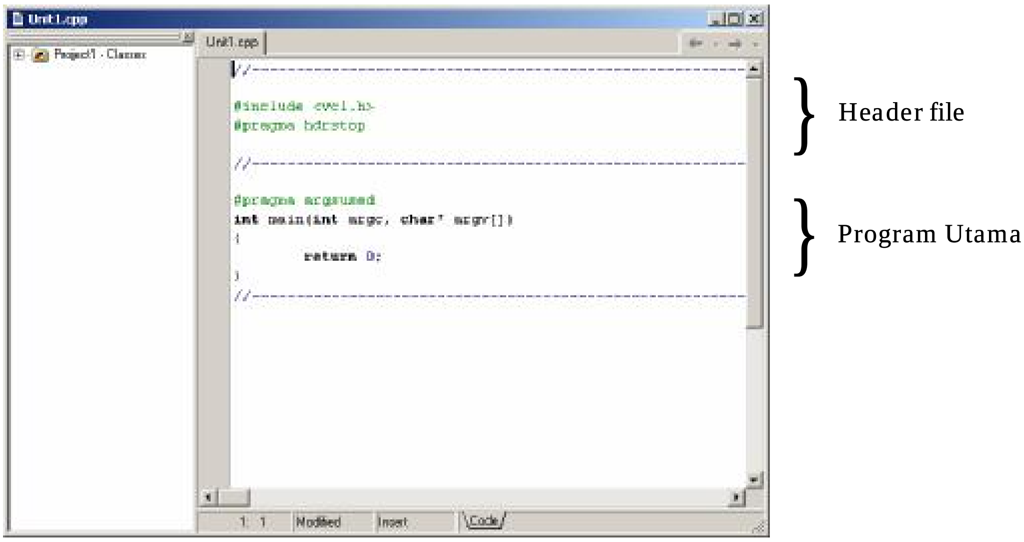
Gambar 4.2 New Items

3. Pilih Tab New, kemudian piih Console Wizard, tekan OK, klik muncul tampilan berikut:



Gambar 4.3 Console Wizard

4. Klik OK untuk memulai membuat aplikasi, akan muncul tampilan awal di bawah ini:



Gambar 4.4 Source Code

5. Untuk menyimpan file, klik menu file, pilih Save All.

   Ada dua file yang harus disimpan

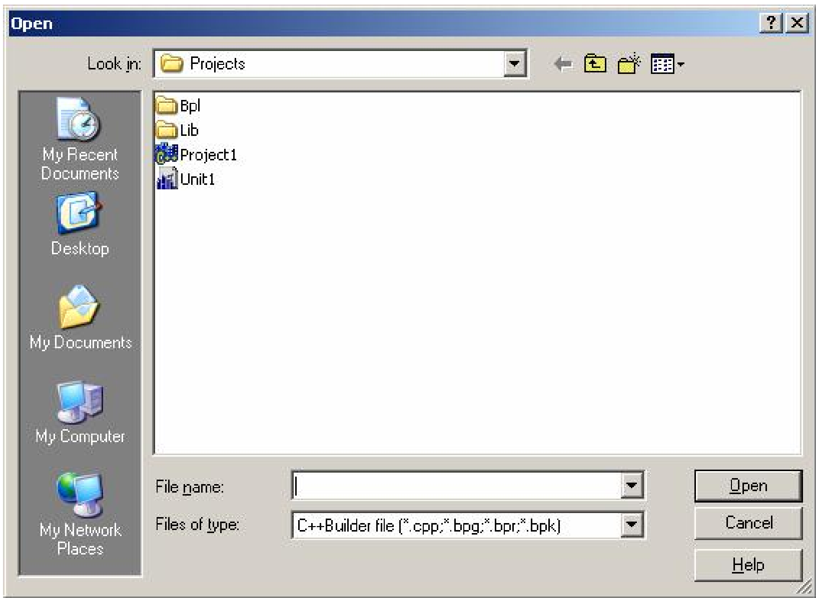
   a. file *.cpp, merupakan file source programnya.

   b. File *.bpr, merupakn file project programnya.

BUKA APLIKASI YANG TELAH ADA

Untuk membuka aplikasi baru, langkah – langkah yang harus dilakukan adalah:

1. Klik menu file, lalu pilih Close all.
2. Klik menu file, lalu pilih open project untuk membuka file projectnya atau piih open file untuk membuka file sourcenya.



Gambar 4.5 Saving

3. Jika memilih open project, cari file *.bpr yang telah disimpan.
4. Jika memilih open file, cari file *.cpp.
5. Klik menu file, pilih Save All, lalu simpan file projectnya.

APLIKASI SEDERHANA – IMPLEMENTASI APLIKASI RUNTUNAN

Berikut merupakan contoh sederhana pemrograman dengan bahasa C++, menggunakan Borland C++ Builder 6.

Kasus: Menghitung Luas persegi panjang.

Luas = panjang x lebar

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>      //untuk mengaktifkan perintah printf dan scanf
#include <conio.h>    //untuk mengaktifkan perintah getch()
```

```
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
       int p,l;                                    } Bagian Deklarasi Variabel
       float Luas;

       printf("Panjang persegi = ");scanf("%d",&p);
       printf("Lebar  persegi = ");scanf("%d",&l);

       Luas = p*l;

       printf("Luas Persegi Panjang = %2.2f",Luas);
       getch();
        return 0;
}
//---------------------------------------------------------------------------
```

Latihan: Buatlah program mencari luas segitiga sama kaki.

5. PreProcessor, Tipe Data, Variabel dan Operator

PREPROCESSOR

#include

PreProcessor - #include digunakan untuk memanggil atau menambahkan file header (file unit) yang telah dibuat sebelumnya. File header berisikan perintah-perintah atau fungsi-fungsi yang telah didefinisikan. Beberapa file header , secara default telah ada di dalam Borland C++ Builder 6, namun bisa dibuat sendiri sesuai dengan kebutuhan. PreProcessor ini berada pada bagian paling atas dari pemrograman. Berikut contoh penulisannya:

#include “iostream.h”
#include “File1.h”

atau

#include <iostream.h>
#include <File1.h>

#define

PreProcessor - #define digunakan untuk mendefiniskan sebuah identifier (pengenal) untuk menggantikan beberapa pernyataan (statement) yang ada di header file. Berikut contoh penulisannya:

#define RI “Republik Indonesia”
#define PLG “Palembang”

artinya, pengenal RI dapat menggantikan posisi pernyataan “Republik Indonesia” dan PLG menggantikan “Palembang”.

#define pi 3.14
#define g 10

Artinya, pi bernilai konstan 3.14 dan g bernilai konstan 10.

Cara pembuatan file header ini sama dengan membuat aplikasi konsol yang baru, namun pada saat memilih jenis aplikasi, pilih icon Header File di kotak New Items seperti di bahwa berikut ini:



Gambar 5.1 Header File 1

File header ini berekstensi *.h. Berikut contoh File Header dengan nama File1.h

```
#define RI "Republik Indonesia"
#define PLG "Palembang"
#define Eol "\n"
#define pi 3.14
#define g 10
```

Simpan dengan nama File1.h, kemudian buat aplikasi konsol baru, lalu masukkan File1.h ini ke dalam source code, seperti di bawah in, simpan dengan nama Unit1.cpp.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>

#include "File1.h"
//---------------------------------------------------------------------------
#pragma argsused
```

```
int main(int argc, char* argv[])
{
	float Luas;
	float Gaya;
	int massa = 10;

	printf(RI Eol);
	printf(PLG Eol);

	Luas = pi*10*10;
	Gaya = massa*g;

	printf("Luas Lingkaran = %2.2f \n",Luas);
	printf("Gaya = %2.2f",Gaya);

	getch();
	return 0;
}
//---------------------------------------------------------------------------
```

TIPE DATA

Tipe data merupakan bagian program yang paling penting karena tipe data mempengaruhi setiap instruksi yang akan dilaksanakan oleh computer. Misalnya saja 5 dibagi 2 bisa saja menghasilkan hasil yang berbeda tergantung tipe datanya. Jika 5 dan 2 bertipe integer maka akan menghasilkan nilai 2, namun jika keduanya bertipe float maka akan menghasilkan nilai 2.5000000. Pemilihan tipe data yang tepat akan membuat proses operasi data menjadi lebih efisien dan efektif.

Dalam bahasa C terdapat lima tipe data dasar, yaitu :

No	Tipe Data	Ukuran	Range	Format	Keterangan
1	char	1 byte	-128 s/d 127	%c	Karakter/string
2	int	2 byte	- 32768 s/d 32767	%i , %d	Integer/bilangan bulat
3	float	4 byte	- 3.4E-38 s/d 3.4E+38	%f	Float/bilangan pecahan
4	double	8 byte	- 1.7E-308 s/d	%lf	Pecahan presisi

			1.7+308		ganda
5	void	0 byte	-	-	Tidak bertipe
6	String	-	-	%s	String

K O N S T A N T A

Konstanta merupakan suatu nilai yang tidak dapat diubah selama proses program berlangsung. Konstanta nilainya selalu tetap. Konstanta harus didefinisikan terlebih dahulu di awal program. Konstanta dapat bernilai integer, pecahan, karakter dan string. Contoh konstanta : 50; 13; 3.14; 4.50005; ‘A’; ‘Bahasa C’. Selain itu, bahasa C juga menyediakan beberapa karakter khusus yang disebut karakter escape, antara lain :

\a : untuk bunyi bell (alert)
\b : mundur satu spasi (backspace)
\f : ganti halaman (form feed)
\n : ganti baris baru (new line)
\r : ke kolom pertama, baris yang sama (carriage return)
\v : tabulasi vertical
\0 : nilai kosong (null)
\’ : karakter petik tunggal
\” : karakter petik ganda
\ \ : karakter garis miring.

V A R I A B E L

Variabel adalah suatu pengenal (identifier) yang digunakan untuk mewakili suatu nilai tertentu di dalam proses program. Berbeda dengan konstanta yang nilainya selalu tetap, nilai dari suatu variable bisa diubah-ubah sesuai kebutuhan. Nama dari suatu variable dapat ditentukan sendiri oleh pemrogram dengan aturan sebagai berikut :

- Terdiri dari gabungan huruf dan angka dengan karakter pertama harus berupa huruf. Bahasa C bersifat case-sensitive artinya huruf besar dan kecil dianggap berbeda. Jadi antara nim, NIM dan Nim dianggap berbeda.
- Tidak boleh mengandung spasi.
- Tidak boleh mengandung symbol-simbol khusus, kecuali garis bawah (underscore). Yang termasuk symbol khusus yang tidak diperbolehkan antara lain : $, ?, %, #, !, &, *, (,), -, +, = dsb
- Panjangnya bebas, tetapi hanya 32 karakter pertama yang terpakai.

Contoh penamaan variabel yang benar :
NIM, a, x, nama_mhs, f3098, f4, nilai, budi, dsb.

Contoh penamaan variable yang salah :
%nilai_mahasiswa, 80mahasiswa, rata-rata, ada spasi, penting!, dsb.

OPERATOR ARITMATIKA

Bahasa C menyediakan lima operator aritmatika, yaitu :

*	: untuk perkalian
/	: untuk pembagian
%	: untuk sisa pembagian (modulus)
+	: untuk pertambahan
-	: untuk pengurangan.

OPERATOR PERBANDINGAN

<	Kurang dari
<=	Kurang dari sama dengan
>	Lebih dari
>=	Lebih dari sama dengan
==	Sama dengan
!=	Tidah sama dengan .

OPERATOR LOGIKA

&&	: Logika AND (DAN)
||	: Logika OR (ATAU)
!	: Logika NOT (INGKARAN).

KODE PENENTU FORMAT

. %c	: Membaca sebuah karakter
. %s	: Membaca sebuah string
. %i, %d	: Membaca sebuah bilangan bulat (integer)
. %f, %e	: Membaca sebuah bilangan pecahan (real)
. %o	: membaca sebuah bilangan octal
. %x	: Membaca sebuah bilangan heksadesimal
. %u	: Membaca sebuah bilangan tak bertanda.

6. Pemilihan

Pemilihan digunakan untuk mengarahkan perjalanan suatu proses. Pemilihan dapat diibaratkan sebagai katup atau kran yang mengatur jalannya air. Bila katup terbuka maka air akan mengalir dan sebaliknya bila katup tertutup air tidak akan mengalir atau akan mengalir melalui tempat lain. Fungsi penyeleksian kondisi penting artinya dalam penyusunan bahasa C, terutama untuk program yang kompleks.

STRUKTUR DASAR PEMILIHAN - IF

Struktur ini digunakan jika kondisinya hanya 1.

```
if (kondisi)
{
        /* aksi */ ;
}
```

Contoh kasus:
Buatlah sebuah program menuliskan teks “Program Diploma Komputer” jika diinput sebuah bilangan ganjil.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    int x;

    printf("Masukkan sebuah bilangan = ");scanf("%d",&x);
```

```
    if(x%2!=0)
    {
        printf("Program Diploma Komputer");
    }
    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

STRUKTUR DASAR PEMILIHAN – IF - ELSE

Struktur ini digunakan jika kondisinya hanya 2.

```
if (kondisi)
{
    /* aksi 1 – jika kondisi terpenuhi*/ ;
}
else
{
    /* aksi 2 – jika kondisi tidak terpenuhi*/
}
```

Contoh kasus:
Buatlah sebuah program menentukan apakah bilangan yang diinput merupakan bilangan ganjil atau genap, dengan asumsi bilangan 0 adalah genap.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
```

```
{
    int x;

    printf("Masukkan sebuah bilangan = ");scanf("%d",&x);

    if(x%2=0)
    {
        printf("%x adalah bilangan genap",x);
    }
    else
    {
        printf("%x adalah bilangan ganjil",x);
    }
    getch();
    return 0;
}
//-----------------------------------------------------------------------
```

S T R U K T U R D A S A R P E M I L I H A N I F – E L S E I F - E L S E

Struktur ini digunakan jika kondisinya lebih dari 2.

```
if (kondisi1)
{
        /* aksi 1 – jika kondisi 1 terpenuhi*/ ;
}
else if (kondisi2)
{
        /* aksi 2 – jika kondisi 1 tidak terpenuhi*/
}
else
{
        /* aksi 3 – jika kondisi 2 tidak terpenuhi*/
}
```

Contoh kasus:
Buatlah sebuah program menentukan bilangan terbesar dari 3 buah bilangan.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
	int a,b,c;

	printf("Masukkan bilangan A = ");scanf("%d",&a);
	printf("Masukkan bilangan B = ");scanf("%d",&b);
	printf("Masukkan bilangan C = ");scanf("%d",&c);

	if(a>=b&&a>=c)
	{
		printf("Bilangan %d terbesar",a);
	}
	else if(b>=a&&b>=c)
	{
		printf("Bilangan %d terbesar",b);
	}
	else
	{
		printf("Bilangan %d terbesar",c);
	}

	getch();
	return 0;
}
//---------------------------------------------------------------------------
```

Latihan: Buatlah sebuah program mencari akar-akar persamaan kuadrat $f(x) = ax^2 + bx + c = 0$, untuk $\boldsymbol{a > 0}$.

STRUKTUR DASAR PEMILIHAN SWITCH – CASE - DEFAULT

Struktur kondisi switch....case....default digunakan untuk penyeleksian kondisi dengan kemungkinan yang terjadi cukup banyak. Struktur ini akan melaksanakan salah satu dari beberapa pernyataan 'case' tergantung nilai kondisi yang ada di dalam switch. Selanjutnya proses diteruskan hingga ditemukan pernyataan 'break'. Jika tidak ada nilai pada case yang sesuai dengan nilai kondisi, maka proses akan diteruskan kepada pernyataan yang ada di bawah 'default'.

Bentuk umum dari struktur ini:

```
switch(variabel)
{
        case 1 : pernyataan-1;
        break;
        case 2 : pernyataan-2;
        break;
        .....
        .....
        case n : pernyataan-n;
        break;
        default : pernyataan-m
}
```

Contoh kasus:
Menentukkan nama hari berdasarkan kode hari.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
```

```
{
    int hari;

    printf("Menentukan nama hari dalam seminggu \n");
    printf("1. Ahad   2. Senin   3. Selasa   4. Rabu   ");
    printf("5. Kamis  6. Jum'at  7. Sabtu \n");
    printf("Kode hari = ");scanf("%d",&hari);

    switch(hari)
    {
        case 1: printf("Hari Ahad");
        break;
        case 2: printf("Hari Senin");
        break;
        case 3: printf("Hari Selasa");
        break;
        case 4: printf("Hari Rabu");
        break;
        case 5: printf("Hari Kamis");
        break;
        case 6: printf("Hari Jum'at");
        break;
        case 7: printf("Hari Sabtu");
        break;
    default: printf("Kode hari SALAH!");
    }

    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

7. Perulangan (Looping)

Dalam bahasa C tersedia suatu fasilitas yang digunakan untuk melakukan proses yang berulangulang sebanyak keinginan kita. Misalnya saja, bila kita ingin menginput dan mencetak bilangan dari 1 sampai 100 bahkan 1000, tentunya kita akan merasa kesulitan. Namun dengan struktur perulangan proses, kita tidak perlu menuliskan perintah sampai 100 atau 1000 kali, cukup dengan beberapa perintah saja. Struktur perulangan dalam bahasa C mempunyai bentuk yang bermacam-macam.

STRUKTUR DASAR PERULANGAN - WHILE

Perulangan WHILE banyak digunakan pada program yang terstruktur. Perulangan ini banyak digunakan bila jumlah perulangannya belum diketahui. Proses perulangan akan terus berlanjut selama kondisinya bernilai benar (true) dan akan berhenti bila kondisinya bernilai salah.

Bentuk Umum:

```
while (kondisi)
{
        /* aksi --- jika kondisi bernilai true (terpenuhi) */
}
```

Contoh Kasus: Menampilkan deret dari 0 – 20.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
```

```
    int i;

    i=0;
    while(i<=20)
    {
        printf("%d ");
        i++;
    }
    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

STRUKTUR DASAR PERULANGAN – DO -WHILE

Pada dasarnya struktur perulangan do....while sama saja dengan struktur while, hanya saja pada proses perulangan dengan while, seleksi berada di while yang letaknya di atas sementara pada perulangan do....while, seleksi while berada di bawah batas perulangan. Jadi dengan menggunakan struktur do…while sekurang-kurangnya akan terjadi satu kali perulangan.

Bentuk Umum:

```
do
{
        /* aksi --- jika kondisi bernilai true (terpenuhi) */
}
while (kondisi);
```

Contoh Kasus: Menampilkan deret menurun dari 20 sampai 0.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
```

```
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
        int i;

        i=0;
        do
        {
                printf("%d ");
                i++;
        }
        while(i<=20);
        getch();
        return 0;
}
//---------------------------------------------------------------------------
```

STRUKTUR DASAR PERULANGAN – FOR

Struktur perulangan for biasa digunakan untuk mengulang suatu proses yang telah diketahui jumlah perulangannya. Dari segi penulisannya, struktur perulangan for tampaknya lebih efisien karena susunannya lebih simpel dan sederhana.

Bentuk Umum:

```
for (inisialisasi; syarat; step)
{
        pernyataan;
}
```

Contoh Kasus: Menampilkan deret bilangan 1 – 100 yang habis dibagi dengan 2 dan habis dibagi dengan 3.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
	int i;

	for(i=1;i<=100;i++)
	{
		if(i%2==0&&i%3==0)
		{
		   printf("%d ",i);
		}
	}
	getch();
	return 0;
}
//---------------------------------------------------------------------------
```

LATIHAN

1. Buatlah Program untuk mencetak tampilan sebagai berikut :

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Gunakan perulangan while atau for..!

2. Buatlah Program untuk mencetak 10 bilangan prima pertama.
2 3 5 7 13 17….

8. Larik (Array)

Array merupakan kumpulan dari nilai-nilai data yang bertipe sama dalam urutan tertentu yang menggunakan nama yang sama. Letak atau posisi dari elemen array ditunjukkan oleh suatu index. Dilihat dari dimensinya array dapat dibagi menjadi Array dimensi satu, array dimensi dua dan array multi-dimensi.

STRUKTUR DASAR LARIK – SATU DIMENSI

Larik satu dimensi merupakan tipe data yang sering digunakan pada pendeklarasian variabel yang sama tapi memiliki indeks yang berbeda, serta pengisian elemen larik dilakukan melalui indeks. Indeks larik secara default dimulai dari 0.

Bentuk umum penulisan:

```
type_data variabel[jumlah_elemen];
```

Contoh:

int data1[7];
int data2[5] = {20,30,10,50,20};

artinya:

Data 1 = Elemen kosong

20	30	10	50	20

Data 2 = Elemen tidak oksong

Contoh program:

```
//-------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
```

```
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
        int x;
        String
hari[7]={"AHAD","SENIN","SELASA","RABU","KAMIS","JUMAT","SABTU"};

        printf("Kode hari (1-7) = ");scanf("%d",&x);
        x=x-1;
        printf("Hari %s",hari[x]);

        getch();
        return 0;
}
//---------------------------------------------------------------------------
```

STRUKTUR DASAR LARIK – DUA DIMENSI

Larik dua dimensi merupakan tipe data yang sering digunakan pada pendeklarasian variabel yang sama tapi memiliki dua indeks yang berbeda, serta pengisian elemen larik dilakukan melalui indeks. Indeks larik secara default dimulai dari 0,0. Jumlah elemennya adalah indeks1 x indeks 2.

Bentuk umum penulisan:

```
type_data variabel[jumlah_elemen1] [jumlah_elemen2];
```

Contoh:

int y[2][2];
int y [2][2]={1,2,3,4};

artinya:

x = Elemen kosong

1	2	3	4

y = Elemen tidak oksong

Contoh kasus: Menghitung jumlah dua matrik A_{2x2} dan B_{2x2}

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
    #define Nmaks 100
    typedef int matrik[Nmaks][Nmaks];
    int n,i,j;
    matrik A,B,C;

    cout<<"Program Perkalian Matrik A 3x3 dan B 3x3\n";
    n = 3;
    gotoxy(7,14);cout<<"Masukkan entri-entri matriks A! \n";
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(7,14+3*i+j);printf("A[%d,%d] = ",i,j);cin>>A[i][j];
            }
        }
    gotoxy(41,14);cout<<"Masukkan entri-entri matriks B! \n";
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(41,14+3*i+j);printf("B[%d,%d] = ",i,j);cin>>B[i][j];
            }
        }
    clrscr();
    cout<<"\n";
    // proses perkalian matrik C = A x B
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            C[i][j]=A[i][1]+B[1][j];
            }
        }
```

```
clrscr();

// proses output matrik A
gotoxy(9,16);
cout<<"A = " ;
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(10+4*j,12+2*i);
            cout<<A[i][j];
            }
        }
// proses output matrik B
gotoxy(9,23);
cout<<"B = " ;
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(10+4*j,19+2*i);
            cout<<B[i][j];
            }
        }
// proses output matrik C
gotoxy(9,30);
cout<<"C = " ;
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(10+4*j,26+2*i);
            cout<<A[i][j];
            }
        }
gotoxy(24,30);
cout<<" x " ;
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            gotoxy(24+4*j,26+2*i);
            cout<<B[i][j];
            }
        }
gotoxy(38,30);
cout<<" = " ;
```

```
        for (i=1;i<=n;i++) {
            for (j=1;j<=n;j++) {
                gotoxy(38+4*j,26+2*i);
                cout<<C[i][j];
                }
            }
        getch();
        return 0;
}
//---------------------------------------------------------------------------
```

Latihan: Menghitung perkalian dua matrik A_{2x2} dan B_{2x2}

STRUKTUR DASAR LARIK – MULTI DIMENSI

Larik multi dimensi merupakan tipe data yang sering digunakan pada pendeklarasian variabel yang sama tapi memiliki lebih dari dua indeks yang berbeda, serta pengisian elemen larik dilakukan melalui indeks. Indeks larik secara default dimulai dari 0,0. Jumlah elemennya adalah indeks1 x indeks 2. x...indeks n

Bentuk umum penulisan:

```
type_data variabel[jumlah_elemen1] [jumlah_elemen2]...[jumlah_elemenn]
```

Contoh:

int x [2][2][2];
int y[2][2[2] ={1,2,3,4,5,6,7,8,};

artinya:

								x = Elemen kosong
1	2	3	4	5	6	7	8	***y = Elemen tidak oksong***

Latihan: Menghitung perkalian dua matrik A_{3x3} dan B_{3x3}.

9. Pointer

PENGERTIAN POINTER

Pointer (variabel penunjuk) adalah suatu variabel yang berisi alamat memori dari suatu variabel lain. Alamat ini merupakan lokasi dari obyek lain (biasanya variabel lain) di dalam memori. Contoh, jika sebuah variabel berisi alamat dari variabel lain, variabel pertama dikatakan menunjuk ke variabel kedua

Operator Pointer ada dua, yaitu :

. Operator &

- Operator & bersifat unary (hanya memerlukan satu operand saja).
- Operator & menghasilkan alamat dari operandnya.

. Operator *

- Operator * bersifat unary (hanya memerlukan satu operand saja).
- Operator * menghasilkan nilai yang berada pada sebuah alamat.

DEKLARASI POINTER

Seperti halnya variabel yang lain, variabel pointer juga harus dideklarasikan terlebih dahulu sebelum digunakan. Bentuk Umum :

Tipe_data *nama_pointer;

Tipe data pointer mendefinisikan tipe dari obyek yang ditunjuk oleh pointer. Secara teknis, tipe apapun dari pointer dapat menunjukkan lokasi (dimanapun) dalam memori. Bahkan operasi pointer dapat dilaksanakan relatif terhadap tipe dasar apapun yang ditunjuk. Contoh, ketika kita mendeklarasikan pointer dengan tipe int*, kompiler akan menganggap alamat yang ditunjuk menyimpan nilai integer - walaupun sebenarnya bukan (sebuah pointer int* selalu menganggap bahwa ia menunjuk ke sebuah obyek bertipe integer, tidak peduli isi sebenarnya). Karenanya, sebelum mendeklarasikan sebuah pointer, pastikan tipenya sesuai dengan tipe obyek yang akan ditunjuk.

Contoh :

```
int *px;
char *sh;
```

Contoh Program :

```
#include "stdio.h"
#include "conio.h"
int  main()
{
        int x, y;             /* x dan y bertipe int */
        int *px;              /* px pointer yang menunjuk objek */
        clrscr();
        x = 87;
        px = &x;          /* px berisi alamat dari x */
        y = *px;          /* y berisi nilai yang ditunjuk px */
        printf("Alamat x = %p \n", &x);
        printf("Isi px = %p \n", px);
        printf("Isi x = %i \n", x);
        printf("Nilai yang ditunjuk oleh px = %i \n", *px);
        printf("Nilai y = %i \n", y);
        getch();
}
```

OPERASI POINTER

Operasi Penugasan

Suatu variable pointer seperti halnya variable yang lain, juga bisa mengalami operasi penugasan. Nilai dari suatu variable pointer dapat disalin ke variable pointer yang lain.

Contoh Program :

```
#include "stdio.h"
#include "conio.h"
int  main()
{
          float *x1, *x2, y;
          clrscr();
          y = 13.45;
          x1 = &y;           /* Alamat dari y disalin ke variabel x1 */
          x2 = x1;           /* Isi variabel x1 disalin ke variabel x2 */
          printf("Nilai variabel y = %.2f ada di alamat %p \n", y, x1);
```

```
        printf("Nilai variabel y = %.2f ada di alamat %p \n", y, x2);
        getch();
}
```

Operasi Aritmatika

Suatu variabel pointer hanya dapat dilakukan operasi aritmatika dengan nilai integer saja. Operasi yang biasa dilakukan adalah operasi penambahan dan pengurangan. Operasi penambahan dengan suatu nilai menunjukkan lokasi data berikutnya (index selanjutnya) dalam memori. Begitu juga operasi pengurangan.

<u>Contoh Program :</u>

```
#include "stdio.h"
#include "conio.h"
int main()
{
        int nilai[3], *penunjuk;
        clrscr();
        nilai[0] = 125;
        nilai[1] = 345;
        nilai[2] = 750;
        penunjuk = &nilai[0];
        printf("Nilai %i ada di alamat memori %p \n", *penunjuk, penunjuk);
        printf("Nilai %i ada di alamat memori %p \n", *(penunjuk+1),
        penunjuk+1);
        printf("Nilai %i ada di alamat memori %p \n", *(penunjuk+2),
        penunjuk+2);
        getch();
}
```

Operasi Logika

Suatu pointer juga dapat dikenai operasi logika.

<u>Contoh Program :</u>

```
#include "stdio.h"
#include "conio.h"
int main()
{
        int a = 100, b = 200, *pa, *pb;
```

```
        clrscr();
        pa = &a;
        pb = &b;
        if(pa < pb)
        printf("pa menunjuk ke memori lebih rendah dari pb\n");
        if(pa == pb)
        printf("pa menunjuk ke memori yang sama dengan pb\n");
        if(pa > pb)
        printf("pa menunjuk ke memori lebih tinggi dari pb\n");
        getch();
}
```

POINTER DAN STRING

Contoh Program 1 :

```
#include "stdio.h"
#include "conio.h"
char *nama1 = "SPIDERMAN";
char *nama2 = "GATOTKACA";
int  main()
{
        char namax;
        clrscr();

        puts("SEMULA :");
        printf("Saya suka >> %s\n", nama1);
        printf("Tapi saya juga suka >> %s\n", nama2);
        /* Penukaran string yang ditunjuk oleh pointer nama1 dan nama2 */
        printf("SEKARANG :");
        printf("Saya suka >> %s\n", nama1);
        printf("Dan saya juga masih suka >> %s\n", nama2);
        getch();
}
```

Contoh Program 2 :

```
#include <stdio.h>
void misteri1(char *);
int main()
{
        char string[] = "characters";
```

```
        printf("String sebelum proses adalah %s", string);
        misteri1(string);
        printf("String setelah proses adalah %s", string);
}

void misteri1(char *s)
{
        while ( *s != '\0' )
        {
                if ( *s >= 'a' && *s <= 'z' )
                *s -= 32;
                ++s;
        }
}
```

P O I N T E R   M E N U N J U K   S U A T U   A R R A Y

Contoh Program :

```
#include "stdio.h"
#include "conio.h"
int  main()
{
        static int tgl_lahir[] = { 13,9,1982 };
        int *ptgl;
        ptgl = tgl_lahir; /* ptgl berisi alamat array */
        printf("Diakses dengan pointer");
        printf("Tanggal = %i\n", *ptgl);
        printf("Bulan = %i\n", *(ptgl + 1));
        printf("Tahun = %i\n", *(ptgl + 2));
        printf("\nDiakses dengan array biasa\n");
        printf("Tanggal = %i\n", tgl_lahir[0]);
        printf("Bulan = %i\n", tgl_lahir[1]);
        printf("Tahun = %i\n", tgl_lahir[2]);
        getch();
}
```

M E M B E R I   N I L A I   A R R A Y   D E N G A N   P O I N T E R

Contoh Program :

```
#include “stdio.h”
#include “conio.h”
int main()
{
        int x[5], *p, k;
        clrscr();
        p = x;
        x[0] = 5; /* x[0] diisi dengan 5 sehingga x[0] = 5 */
        x[1] = x[0]; /* x[1] diisi dengan x[0] sehingga x[1] = 5 */
        x[2] = *p + 2; /* x[2] diisi dengan x[0] + 2 sehingga x[2] = 7 */
        x[3] = *(p+1) – 3; /* x[3] diisi dengan x[1] - 3 sehingga x[3] = 2 */
        x[4] = *(x + 2); /* x[4] diisi dengan x[2] sehingga x[4] = 7 */
        for(k=0; k<5; k++)
        printf(“x[%i] = %i \n”, k, x[k]);
        getch();
}
```

POINTER DAN RECORD

Latihan6.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
#include <string.h>

struct biodata
{
    char nim[11];
    char nama[25];
    struct
    {
        char jalan[20];
        char kota[15];
        char kodepos[5];
    } alamat;
    struct
```

```
	{
		int tanggal;
		int bulan;
		int tahun;
	} lahir;
};
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
	struct biodata *mahasiswa;
	/* Input Biodata Mahasiswa*/
	printf("NIM                    : ");scanf("%s",mahasiswa->nim);
	printf("Nama Mahasiswa  : ");scanf("%s",mahasiswa->nama);
	printf("\n");
	printf("Alamat\n");
	printf("Jalan                  : ");scanf("%s",mahasiswa->alamat.jalan);
	printf("Kota                   : ");scanf("%s",mahasiswa->alamat.kota);
	printf("Kode Pos               : ");scanf("%s",mahasiswa->alamat.kodepos);
	printf("\n");
	printf("Lahir\n");
	printf("Tanggal(xx)            : ");scanf("%i",&mahasiswa->lahir.tanggal);
	printf("Bulan(xx)              : ");scanf("%i",&mahasiswa->lahir.bulan);
	printf("Tahun(xxxx)            : ");scanf("%i",&mahasiswa->lahir.tahun);
	printf("\n\n");

	/* Output Biodata Mahasiswa*/
	printf("NIM                 : %s\n",mahasiswa->nim);
	printf("Nama                : %s\n",mahasiswa->nama);
	printf("Alamat              : Jl. %s %s %s\n",mahasiswa->alamat.jalan,
				mahasiswa->alamat.kota,
				mahasiswa->alamat.kodepos);
	printf("Tanggal Lahir  : %i - %i - %i  \n",mahasiswa->lahir.tanggal,
				mahasiswa->lahir.bulan,mahasiswa->lahir.tahun);
	getch();
	return 0;
}
//---------------------------------------------------------------------------
```

10. Fungsi dan Prosedur

FUNGSI

Pengertian Fungsi

Fungsi merupakan suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya terpisah dari program yang memanggilnya. Fungsi merupakan elemen utama dalam bahasa C karena bahasa C sendiri terbentuk dari kumpulan fungsi-fungsi. Dalam setiap program bahasa C, minimal terdapat satu fungsi yaitu fungsi main(). Fungsi banyak diterapkan dalam program-program C yang terstruktur. Keuntungan penggunaan fungsi dalam program yaitu program akan memiliki struktur yang jelas (mempunyai readability yang tinggi) dan juga akan menghindari penulisan bagian program yang sama. Dalam bahasa C fungsi dapat dibagi menjadi dua, yaitu fungsi pustaka atau fungsi yang telah tersedia dalam C++ dan fungsi yang didefinisikan atau dibuat oleh programmer.

Beberapa Fungsi yang telah tersedia di dalam bahasa C++, antara lain:

Fungsi Operasi String (tersimpan dalam header file "string.h")

- . strcpy()
  Berfungsi untuk menyalin suatu string asal ke variable string tujuan.
  Bentuk umum : strcpy(var_tujuan, string_asal);
- . strlen()
  berfungsi untuk memperoleh jumlah karakter dari suatu string.
  Bentuk umum : strlen(string);
- . strcat()
  Digunakan untuk menambahkan string sumber ke bagian akhir dari string tujuan.
  Bentuk umum : strcat(tujuan, sumber);
- . strupr()
  Digunakan untuk mengubah setiap huruf dari suatu string menjadi huruf capital.
  Bentuk umum : strupr(string);

. strlwr()

Digunakan untuk mengubah setiap huruf dari suatu string menjadi huruf kecil semua.

Bentuk umum : strlwr(string);

. strcmp()

Digunakan untuk membandingkan dua buah string.

Hasil dari fungsi ini bertipe integer dengan nilai :

(a) Negative, jika string pertama kurang dari string kedua.

(b) Nol, jika string pertama sama dengan string kedua

(c) Positif, jika string pertama lebih besar dari string kedua.

Bentuk umum : strcmp(string1, string2);

Fungsi Operasi Karakter (tersimpan dalam header "ctype.h")

. islower()

Fungsi akan menghasilkan nilai benar (bukan nol) jika karakter merupakan huruf kecil.

Bentuk umum : islower(char);

. isupper()

Fungsi akan menghasilkan nilai benar (bukan nol) jika karakter merupakan huruf kapital.

Bentuk umum : isupper(char);

. isdigit()

Fungsi akan menghasilkan nilai benar (bukan nol) jika karakter merupakan sebuah digit.

Bentuk umum : isdigit(char);

. tolower()

Fungsi akan mengubah huruf capital menjadi huruf kecil.

Bentuk umum : tolower(char);

. toupper()

Fungsi akan mengubah huruf kecil menjadi huruf kapital.

Bentuk umum : toupper(char);

Fungsi Operasi Matematik (tersimpan dalam header "math.h" dan "stdlib.h")

. sqrt()

Digunakan untuk menghitung akar dari sebuah bilangan.

Bentuk umum : sqrt(bilangan);

. pow()

Digunakan untuk menghitung pemangkatan suatu bilangan.

Bentuk umum : pow(bilangan, pangkat);

. sin(), cos(), tan()

Masing-masing digunakan untuk menghitung nilai sinus, cosinus dan tangens dari suatu sudut.

Bentuk umum :

sin(sudut);

cos(sudut);

tan(sudut);

. atof()

Digunakan untuk mengkonversi nilai string menjadi bilangan bertipe double.

Bentuk umum : atof(char x);

. atoi()

Digunakan untuk mengkonversi nilai string menjadi bilangan bertipe integer.

Bentuk umum : atoi(char x);

. div()

Digunakan untuk menghitung hasil pembagian dan sisa pembagian.

Bentuk umum : div_t div(int x, int y)

Strukturnya :

```
typedef struct
{ int qout;            // hasil pembagian
      int rem          // sisa pembagian
} div_t;
```

. max()

Digunakan untuk menentukan nilai maksimal dari dua buah bilangan.

Bentuk umum : max(bilangan1, bilangan2);

. min()

Digunakan untuk menentukan bilangan terkecil dari dua buah bilangan.

Bentuk umum : min(bilangan1, bilangan2);

Membuat Fungsi Sendiri

Deklarasi Fungsi

Sebelum digunakan (dipanggil), suatu fungsi harus dideklarasikan dan didefinisikan terlebih dahulu. Bentuk umum pendeklarasian fungsi adalah :

tipe_fungsi nama_fungsi(parameter_fungsi);

Sedangkan bentuk umum pendefinisian fungsi adalah :

```
Tipe_fungsi nama_fungsi(parameter_fungsi)
{
          statement
          statement
          ………...
          ………...
}
```

Hal-hal yang perlu diperhatikan dalam penggunaan fungsi :

1. Kalau tipe fungsi tidak disebutkan, maka akan dianggap sebagai fungis dengan nilai keluaran bertipe integer.
2. Untuk fungsi yang memiliki keluaran bertipe bukan integer, maka diperlukan pendefinisian penentu tipe fungsi.
3. Untuk fungsi yang tidak mempunyai nilai keluaran maka dimasukkan ke dalam tipe void
4. Pernyataan yang diberikan untuk memberikan nilai akhir fungsi berupa pernyataan return.
5. Suatu fungsi dapat menghasilkan nilai balik bagi fungsi pemanggilnya.

Parameter Formal dan Parameter Aktual

1. Parameter Formal adalah variabel yang ada pada daftar parameter dalam definisi fungsi.
2. Parameter Aktual adalah variabel (parameter) yang dipakai dalam pemanggilan fungsi.

Cara Melewatkan Parameter

Cara melewatkan suatu parameter dalam Bahasa C ada dua cara yaitu :

1. Pemanggilan Secara Nilai (***Call by Value***)
   - Call by value akan menyalin nilai dari parameter aktual ke parameter formal.
   - Yang dikirimkan ke fungsi adalah nilai dari datanya, bukan alamat memori letak dari datanya.
   - Fungsi yang menerima kiriman nilai akan menyimpannya di alamat terpisah dari nilai aslinya yang digunakan oleh bagian program yang memanggil fungsi.
   - Perubahan nilai di fungsi (parameter formal) tidak akan merubah nilai asli di bagian program yang memanggilnya.

- Pengiriman parameter secara nilai adalah pengiriman searah, yaitu dari bagian program yang memanggil fungsi ke fungsi yang dipanggil.
- Pengiriman suatu nilai dapat dilakukan untuk suatu ungkapan, tidak hanya untuk sebuah variabel, elemen array atau konstanta saja.

2. Secara Referensi (***Call by Reference***)
   - Pemanggilan secara Referensi merupakan upaya untuk melewatkan alamat dari suatu variabel ke dalam fungsi.
   - Yang dikirimkan ke fungsi adalah alamat letak dari nilai datanya, bukan nilai datanya.
   - Fungsi yang menerima kiriman alamat ini makan menggunakan alamat yang sama untuk mendapatkan nilai datanya.
   - Perubahan nilai di fungsi akan merubah nilai asli di bagian program yang memanggil fungsi.
   - Pengiriman parameter secara referensi adalah pengiriman dua arah, yaitu dari fungsi pemanggil ke fungsi yang dipanggil dan juga sebaliknya.
   - Pengiriman secara acuan tidak dapat bdilakukan untuk suatu ungkapan.

Penggolongan Variabel berdasarkan Kelas Penyimpanan (Storage Class)

1. Variabel lokal
   Variabel lokal adalah variabel yang dideklarasikan di dalam fungsi.
   Sifat-sifat variabel lokal :
   - Secara otomatis akan diciptakan ketika fungsi dipanggil dan akan lenyap ketika proses eksekusi terhadap fungsi berakhir.
   - Hanya dikenal oleh fungsi tempat variabel dideklarasikan.
   - Tidak ada inisialisasi secara otomatis (saat variabel diciptakan nilainya random).
   - Dideklarasikan dengan menambahkan kata “auto” (opsional).
2. Variabel global (eksternal)
   Variabel global (eksternal) adalah variabel yang dideklarasikan di luar fungsi.
   Sifat-sifat variabel global :
   - Dikenal (dapat diakses) oleh semua fungsi.
   - Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
   - Dideklarasikan dengan menambahkan kata “extern” (opsional).

3. Variabel Statis

   Variabel statis adalah variabel yang nilainya tetap dan bisa berupa variabel lokal (internal) dan variabel global (eksternal).

   Sifat-sifat variabel statis :

   - Jika bersifat internal (lokal), maka variabel hanya dikenal oleh fungsi tempat variabel dideklarasikan.
   - Jika bersifat eksternal (global), maka variabel dapat dipergunakan oleh semua fungsi yang terletak pada program yang sama.
   - Nilai variabel statis tidak akan hilang walau eksekusi terhadap fungsi telah berakhir.
   - Inisialisasi hanya perlu dilakukan sekali saja, yaitu pada saat fungsi dipanggil pertama kali.
   - Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
   - Dideklarasikan dengan menambahkan kata “static”.

4. Variabel Register

   Variabel Register adalah variabel yang nilainya disimpan dalam resister dan bukan dalam memori RAM.

   Sifat-sifat variabel register :

   - Hanya dapat diterapkan pada variabel lokal yang bertipe int dan char.
   - Digunakan untuk mengendalikan proses perulangan (looping).
   - Proses perulangan akan lebih cepat karena variabel register memiliki kecepatan yang lebih tinggi dibandingkan variabel biasa.
   - Dideklarasikan dengan menambahkan kata “register”.

FUNGSI REKURSIF

Rekursif ialah salah satu teknik pemrograman dengan cara memanggil sebuah fungsi dari dirinya sendiri, baik itu secara langsung maupun tidak langsung. Pemanggilan fungsi rekursif secara langsung berarti dalam fungsi tersebut terdapat ***statement*** untuk memanggil dirinya sendiri sedangkan secara tidak langsung berarti fungsi rekursif tersebut memanggil 1 atau lebih fungsi lain sebelum memanggil dirinya sendiri.

```
Tipe_fungsi
nama_fungsi([parameter])
{
     // statement
     coba([parameter])
     // statement
}
```

Fungsi Rekursif Langsung

```
[ret-val] satu([parameter])
{
     // statement
     dua([parameter])
     // statement
}
```

```
[ret-val] dua([parameter])
{
     // statement
     satu([parameter])
     // statement
}
```

Fungsi Rekursif Tak Langsung

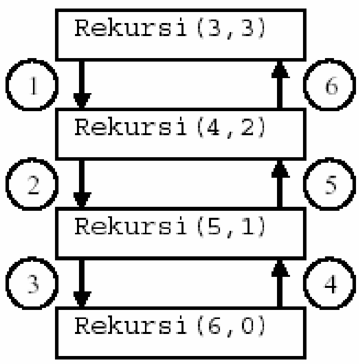
Rekursif tidak selalu lebih jelek daripada iteratif . Ada kalanya sebuah fungsi rekursif justru empermudah penyelesaian masalah yang ditemui pada kasus iteratif (pengulangan). Kelemahan pada proses rekursif antar lain, memerlukan tempat penampungan stack yang cukup besar. Karena setiap kali pemanggilan fungsi , register – register seperti cs (untuk memory far) dan ip harus disimpan , belum lagi untuk penanganan local variable serta parameter fungsi yang tentunya membutuhkan ruang untuk stack lebih banyak lagi. Selain itu karena setiap pemanggilan fungsi , register dan memory harus di push ke stack maka setelah selesai pemanggilan perlu diadakannya pop stack. untuk mengembalikan memory dan register kembali ke keadaan awal , ini sering disebut sebagai ***overhead*** .

Proses Rekursif

Untuk dapat memahami proses yang terjadi dalam sebuah fungsi rekursif , marilah kita simak contoh fungsi rekursif berikut :

```
void rekursi(int a,int b)
{
   if (b == 0) return;
   a++;
   b--;
   printf("Masuk -> a = %d || b = %d \n",a,b);
   rekursi(a,b);
   printf("Keluar -> a = %d || b = %d \n",a,b);
}
```

Misalkan Fungsi tersebut dipanggil dengan nilai a = 3 dan b = 3 maka pertama tama di cek apakah b = 0 (if (b == 0) return), jika sama maka keluar. Ternyata nilai b tidak sama dengan 0 maka tambahkan a dengan 1 dan kurangi b dengan 1 . Maka keadaan sekarang menjadi a = 4 dan b = 2 . Baris berikutnya menampilkan nilai a dan b ke layar (printf("Masuk -> a = %d || b = %d \n",a,b)). Kemudian panggil fungsi rekursi dengan nilai a = 4 dan b = 2 . Langkah – langkah tersebut diulang terus sampai pemanggilan fungsi rekursi dengan nilai a = 6 dan b = 0. Pada saat ini kondisi if bernilai benar sehingga fungsi akan keluar (return) dan melanjutkan perintah setelah pemanggilan fungsi rekursi dengan a = 6 dan b = 0. Yaitu mencetak nilai a dan b (printf("Keluar -> a = %d || b = %d \n",a,b)). Setelah mencetak nilai a dan b maka fungsi rekursif akan keluar lagi , dan melanjutkan perintah setelah pemanggilan fungsi rekursif sebelumnya dimana nilai a = 5 dan b = 1 . Demikian seterusnya sampai nilai a = 4 dan nilai b = 2. yang tidak lain pemanggilan fungsi rekurif yang pertama. Proses pemanggilan fungsi rekursif dapat diilustrasikan :



Langkah ke :

1. a = 4 ; b = 2 . Cetak : Masuk -> a = 4 || b = 2
2. a = 5 ; b = 1 . Cetak : Masuk -> a = 5 || b = 1
3. a = 6 ; b = 0 . Cetak : Masuk -> a = 6 || b = 0
4. a = 6 ; b = 0 , Cetak : Keluar -> a = 6 || b = 0
5. a = 5 ; b = 1 , Cetak : Keluar -> a = 5 || b = 1
6. a = 4 ; b = 2 , Cetak : Keluar -> a = 4 || b = 2

Untuk memperjelas lagi penggunaan fungsi rekursif dibawah ini akan di berikan contoh-contoh program dengan menggunakan rekursif .

Menghitung Nilai Faktorial Dengan Rekursif

Untuk menghitung nilai faktorial bilangan bulat positif marilah kita daftarkan dulu nilai – nilai faktorial untuk mempermudah pengambilan algoritma .

$$\left.\begin{array}{l} 0! = 1 \\ 1! = 1 \\ 2! = 2 \times 1 \\ 3! = 3 \times 2 \times 1 = 3 \times 2! \\ 4! = 4 \times 3 \times 2 \times 1 = 4 \times 3! \end{array}\right\} \quad N! = N \times (N-1)\,!$$

Dari daftar diatas dapat dibuat fungsi rekursi untuk menghitung nilai faktorial ke-n yaitu:

$$Fakt(n) = \begin{cases} n > 1 \rightarrow n * Fakt(n-1) \\ n == 0 \parallel n == 1 \rightarrow 1 \end{cases}$$

Jika di terjemahkan ke dalam bahasa C menjadi :

```
int Fakt(int n)
{
    if (n == 1 || n == 0) return 1;
    return n * Fakt(n-1);
}
```

PROSEDUR

Pengertian Prosedur

Seperti halnya dan Fungsi, Prosedur merupakan suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya terpisah dari program yang memanggilnya. Perbedaanya adalah, Fungsi juga dideklarasikan seperti variabel, dan mengembalikan nilai fungsi ke pemanggilnya. Sedangkan Prosedur tidak dideklarasikan seperti variabel dan tidak mengembalikan nilai tertentu, tapi dapat memberikan hasil yang ada di dalam prosedur itu. Prosedur juga sering disebut dengan fungsi tanpa tipe fungsi, artinya dengan menggunakan tipe void.

Contoh Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>

float Volume;          /* Variabel global */
```

```
void LuasLingkaran()    /* deklarasi fungsi tanpa parameter */
{
      int r;                   /* variabel lokal */
      float Luas;              /* variabel lokal */

      r=10;
      Luas = 3.14*r*r;
      printf("Luas Lingkaran = %2.2f",Luas);
}

void VolumeBalok(int &p,int &l,int &t)    /* deklarasi fungsi dengan parameter */
{
      Volume = p*l*t;
}
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
      LuasLingkaran();              /* Pemanggilan Fungsi tanpa parameter */
      VolumeBalok(2,3,4);           /* Pemanggilan fungsi dengan parameter */
      printf("\nVolume Balok = %2.2f",Volume);

      getch();
      return 0;
}
//---------------------------------------------------------------------------
```

11. Operasi File

File adalah sebuah organisasi dari sejumlah record. Masing-masing record bisa terdiri dari satu atau beberapa field. Setiap field terdiri dari satu atau beberapa byte.

MEMBUKA FILE

Untuk membuka atau mengaktifkan file, fungsi yang digunakan adalah fungsi fopen(). File dapat berupa file biner atau file teks. File biner adalah file yang pola penyimpanan di dalam disk dalam bentuk biner, yaitu seperti bentuk pada memori (RAM) computer. File teks adalah file yang pola penyimpanan datanya dalam bentuk karakter. Penambahan yang perlu dilakukan untuk menentukan mode teks atau biner adalah "t" untuk file teks dan "b" untuk file biner. Prototype fungsi fopen() ada di header fungsi "stdio.h"

Bentuk umum :

```
file *fopen(char *namafile, char *mode);
```

Keterangan :

- namafile adalah nama dari file yang akan dibuka/diaktifkan.
- mode adalah jenis operasi file yang akan dilakukan terhadap file.

Jenis-jenis operasi file :

- ***r*** : menyarakan file hanya dapat dibaca (file harus sudah ada)
- w : menyatakan file baru akan dibuat/diciptakan (file yang sudah ada akan dihapus)
- ***a*** : untuk membuka file yang sudah ada dan akan dilakukan proses penambahan data (jika file belum ada, otomatis akan dibuat)
- ***r+*** : untuk membuka file yang sudah ada dan akan dilakukan proses pembacaan dan penulisan.
- ***w+*** : untuk membuka file dengan tujuan untuk pembacaan atau penulisan.

Jika file sudah ada, isinya akan dihapus. a+ : untuk membuka file, dengan operasi yang akan dilakukan berupa perekaman maupun pembacaan. Jika file sudah ada, isinya akan dihapus.

Contoh :

```
pf = fopen("COBA.TXT", "w");
```

MENUTUP FILE

Untuk menutup file, fungsi yang digunakan adalah fclose(). Prototype fungsi fclose() ada di header file “stdio.h”

Bentuk Umum :

```
int fclose(FILE *pf);
```

atau

```
int fcloseall(void);
```

MELAKSANAKAN PROSES FILE

Menulis Karakter

Untuk menulis sebuah karakter, bentuk yang digunakan adalah :

```
putc(int ch, file *fp)
```

Ket:

- fp adalah pointer file yang dihasilkan oleh fopen()
- ch adalah karakter yang akan ditulis.

Membaca Karakter

Untuk membaca karakter dari file, fungsi yang digunakan adalah :

```
getc(file *fp);
```

Ket:

- fp adalah pointer file yang dihasilkan oleh fopen()
- Fungsi feof(), digunakan untuk mendeteksi akhir file pada saat membaca data foef(file *fp).

Contoh Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
#define CTRL_Z 26
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    FILE *pf;           /* pointer ke file */
```

```
    char kar;

    if((pf = fopen("COBA.TXT", "w")) == NULL) /* ciptakan file */
    {
        cputs("File tak dapat diciptakan !\r\n");
        exit(1);        /* selesai */
    }
    while((kar=getche()) != CTRL_Z)
    putc(kar, pf);          /* tulis ke file */
    fclose(pf);             /* tutup file */

    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

Membaca dan Menulis String

Fungsi untuk membaca dan menulis string adalah : fgets() dan fputs()

Bentuk Umum :

```
fgets(char *str, int p, file *fp)
fputs(char *str, file *fp)
```

Membaca dan Menulis Blok Data

Fungsi untuk membaca dan menulis blok data adalah : fread() dan fwrite()

Bentuk umum :

```
fread(void *buffer, int b_byte, int c, file *fp);
fwrite(void *buffer, int b_byte, int c, file *fp);
```

Keterangan :

- buffer adalah pointer ke sebuah area di memori yang menampung data yang akan dibaca dari file.
- b_byte adalah banyaknya byte yang akan dibaca atau ditulis ke file
- c adalah banyaknya item dibaca/ditulis.

Membaca dan Menulis File yang Terformat

Jika diinginkan, data bilangan dapat disimpan ke dalam file dalam keadaan terformat.

Fungsi yang digunakan adalah :

```
fprintf(ptr_file, “string control”, daftar argument);
```

fscanf(pts_file, “string control”, daftar argument);

Contoh bagian deklarasi :

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>

typedef struct {
        long NIM;
        char Nama[25];
        float IPK;
        } DataMhs;
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
     FILE *MHS;                     /* File data record */
     DataMhs RekMhs;

     FILE *BIL;
     int i;                          /* File data integer */


     FILE *KAR;
     char c;                        /* File data karakter */

     getch();
     return 0;
}
//---------------------------------------------------------------------------
```

Contoh program membuat file baru:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include <stdio.h>
#include <conio.h>

typedef struct { long NIM;
           char Nama[25];
           float IPK;
         } DataMhs;
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
     FILE *MHS;
     DataMhs RekMhs;

     MHS = fopen("kuliah.dat","w");
     printf("NIM  = ");scanf("%s",&RekMhs.NIM);
     printf("Nama = ");scanf("%s",&RekMhs.Nama);
     printf("IPK  = ");scanf("%f",&RekMhs.IPK);

     fwrite(&RekMhs, sizeof(RekMhs),1,MHS);
     fclose(MHS);

     getch();
     return 0;
}
//---------------------------------------------------------------------------
```

Contoh program membuka dan membaca file yang ada:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
typedef struct { long NIM;
           char Nama[25];
           float IPK;
         } DataMhs;
```

```
//------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
        /* Tulis file ke file kuliah.dat*/
        FILE *MHS;
        DataMhs RekMhs;
        printf("Tulis ke file kuliah.dat\n");
        MHS = fopen("kuliah.dat","w");

        printf("NIM  = ");scanf("%ld",&RekMhs.NIM);
        printf("Nama = ");scanf("%s",&RekMhs.Nama);
        printf("IPK  = ");scanf("%f",&RekMhs.IPK);

        fwrite(&RekMhs, sizeof(RekMhs),1,MHS);
        fclose(MHS);

        /* Baca file dari file kuliah.dat*/
        printf("Buka File kuliah.dat\n");
        FILE *BukaMhs;

        BukaMhs = fopen("kuliah.dat","r");
        fread(&RekMhs, sizeof(RekMhs),1,BukaMhs);

        printf("NIM  = %ld \n",RekMhs.NIM);
        printf("Nama = %s \n",RekMhs.Nama);
        printf("IPK  = %2.2f \n",RekMhs.IPK);
        fclose(BukaMhs);

        getch();
        return 0;
}
//------------------------------------------------------------------------
```

MELAKSANAKAN PROSES FILE

File sekuensial berisi rekord-rekord data yang tidak mengenal posisi baris atau nomor rekord pada saat aksesnya, dan setiap record dapat mempunyai lebar yang berbeda-beda. Akses terhadapnya selalu dimulai dari awal file dan berjalan satu

persatu menuju akhir dari file. Dengan demikian, penambahan file hanya dapat dilakukan terhadap akhir file, dan akses terhadap baris tertentu harus dimulai dari awal file. Fungsi baku yang terkait dengan file sekuensial ini antara lain adalah fprintf,fscanf, dan rewind.

Program berikut menyajikan penanganan file sekuensial tentang data nasabah yang berisi tiga field, yaitu nomor identitas (***account***), nama (***name***), dan posisi tabungannya (***balance***) untuk (1) menyajikan yang tabungannya bernilai nol, (2) berstatus kredit, dan (3) berstatus debet. File data tersimpan dengan nama klien.dat.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
      int request, account;
      float balance;
      char name[25];
      FILE *cfPtr;

      if ( (cfPtr = fopen("klien.dat", "r+") ) == NULL )
      {
            printf("File could not be opened\n");
      }
      else
      {
            printf ( "Enter request\n"
            "1 - List accounts with zero balances\n"
            "2 - List accounts with credit balances\n"
            "3 - List accounts with debit balances\n"
            "4 - End of run\n? " ) ;
            scanf( "%d", &request );
            while (request != 4)
            {
                  fscanf (cfPtr, "%d%s%f", &account, name, &balance);
```

```
        switch (request)
        {
        case 1:
            printf ("\nAccounts with zero balances:\n");
            while ( !feof(cfPtr) )
            {
                if (balance == 0)
                printf ("%-10d%-13s7.2f\n", account, name, balance);
                fscanf (cfPtr, "%d%s%f", &account, name, &balance);
            }
        break;
        case 2:
            printf ("\nAccounts with credit balances:\n");
            while ( !feof(cfPtr) )
            {
                if (balance < 0)
                printf ("%-10d%-13s7.2f\n", account, name, balance);
                fscanf (cfPtr, "%d%s%f", &account, name, &balance);
            }
        break;
        case 3:
            printf ("\nAccounts with debit balances:\n");
            while ( !feof(cfPtr) )
            {
                if (balance > 0)
                printf ("%-10d%-13s7.2f\n", account, name, balance);
                fscanf (cfPtr, "%d%s%f", &account, name, &balance);
            }
        break;
        }
        rewind(cfPtr);
        printf( "\n? ");
        scanf ("%d", &request);
    }
  }
  getch();
  return 0;
}
//------------------------------------------------------------------------
```

12. Class Dasar C++

Pemrograman C++ memerlukan pemahaman yang memadai untuk menterjemahkan desain ke dalam bentuk implementasi, terutama untuk desain yang menggunakan abstraksi class. Fokus pembahasan pada aspek pembentukan obyek (***construction***) sebuah class, dan proses sebaliknya pada saat obyek tersebut sudah tidak digunakan lagi (***destruction***).

DEKLARASI DAN DEFINISI

Deklarasi dan definisi adalah langkah awal dalam setiap penulisan program tidak terkecuali dalam bahasa C++. Deklarasi dan definisi diperlukan untuk semua tipe data termasuk tipe data bentukan user (***user-defined type***).
Bentuk sederhana deklarasi class adalah sebagai berikut,

```
class C {    }; atau
struct C {    };
```

dalam bahasa C++ struct dan class mempunyai pengertian yang sama. Deklarasi class dengan struct mempunyai anggota dengan akses public kecuali jika dinyatakan lain.

```
struct C
{
        int i;
        void f();
}
class C
{
        public:
        int i;
        void f();
}
```

Kedua deklarasi tersebut mempunyai arti yang sama. Hal ini adalah pilihan desain yang diambil oleh desainer C++ (Bjarne Stroustrup) untuk menggunakan C sebagai basis C++ ketimbang membuat bahasa yang sama sekali baru. Tentunya ada

konsekuensi atas pilihan desain ini, salah satu contoh adalah kompatibilitas terhadap bahasa C. Dalam bahasa C deklarasi,

```
struct C { … };
```

menyatakan C sebagai nama tag. Nama tag berbeda dengan nama tipe, sehingga C (nama tag) tidak dapat dipergunakan dalam deklarasi yang membutuhkan C sebagai suatu tipe obyek. Kedua contoh deklarasi berikut ini tidak valid dalam bahasa C,

```
C c;     /* error, C adalah nama tag */
C *pc;   /* error, C adalah nama tag */
```

Dalam bahasa C, kedua deklarasi tersebut harus ditulis sebagai berikut,

```
struct C c;
struct C *pc;
```

atau menggunakan typedef sebagai berikut,

```
struct C { … };
typedef struct C C;
C c;
C *pc;
```

C++ memperlakukan nama class, C sebagai nama tag sekaligus nama tipe dan dapat dipergunakan dalam deklarasi. Kata class tetap dapat dipergunakan dalam deklarasi, seperti contoh berikut ini,

```
class C c;
```

Dengan demikian C++ tidak membedakan nama tag dengan nama class, paling tidak dari sudut pandang pemrogram (***programmer***), dan tetap menerima deklarasi ***structure*** seperti dalam bahasa C. Kompatibilitas C++ terhadap tidak sebatas perbedaan nama tag dan nama tipe, karena standar C++ masih perlu mendefinisikan tipe POD (Plain Old Data). POD type mempunyai banyak persamaan dengan structure dalam C. Standar C++ mendefinisikan POD type sebagai obyek suatu class yang tidak mempunyai userdefined constructor, anggota protected maupun private, tidak punya base class, dan tidak memiliki fungsi virtual. Dalam desain suatu aplikasi terdiri atas banyak class, dan masing-masing class tidak berdiri sendiri melainkan saling bergantung atau berhubungan satu sama lain. Salah satu contoh hubungan tersebut adalah hubungan antara satu class dengan satu atau lebih ***base***

class atau ***parent class***. Jika class C mempunyai base class B, dikenal dengan ***inheritance***, maka deklarasi class menjadi,

```
class C : public B {}; atau
class C : protected B {}; atau
class C : private B {};
```

akses terhadap anggota base class B dapat bersifat public, protected, maupun private,atau disebut dengan istilah ***public***, ***protected*** atau ***private inheritance***. Class C disebut dengan istilah ***derived class***. Jika tidak dinyatakan bentuk akses secara eksplisit, seperti dalam deklarasi berikut:

```
class C : B
```

maka interpretasinya adalah private inheritance (***default***), tetapi jika menggunakan struct maka tetap merupakan public inheritance. Jika desainer class C tersebut menginginkan hubungan ***multiple inheritance*** (MI) terhadap class B dan A, maka deklarasi class C menjadi,

```
class C : public B, public A { };
```

Sebuah class, seperti halnya class C mempunyai anggota berupa data maupun fungsi (***member function***). Isi class tersebut berada diantara tanda kurung { } dan dipilah-pilah sesuai dengan batasan akses yang ditentukan perancang (desainer) class tersebut.

```
class C : public B
{
        public:
        (explicit) C()(:member-initializer);
        C(const C& );
        C& operator=(const C&);
        (virtual)~C();
        statement lain
        (protected: statement)
        (private: statement)
};
```

Secara ringkas batasan akses (***access specifiers***) mempunyai arti seperti ditunjukkan pada table berikut ini,

Batasan Akses	Arti
Public	Semua class atau bebas
Protected	Class itu sendiri, ***friend***, atau ***derived class***
Private	Class itu sendiri, ***friend***

Sebuah class dapat memberikan ijin untuk class lain mengakses bagian protected maupun private class tersebut melalui hubungan ***friendship*** (dinyatakan dengan ***keyword*** friend). Sebuah class mempunyai beberapa fungsi khusus, yaitu ***constructor***, ***copy constructor***, ***destructor*** dan ***copy assignment operator***.

Constructor

C() adalah anggota class yang bertugas melakukan inisialisasi obyek (***instance***) dari suatu class C. Constructor mempunyai nama yang sama dengan nama class, dan tidak mempunyai ***return value***. Sebuah class dapat mempunyai lebih dari satu constructor. Constructor yang tidak mempunyai argumen, disebut ***default constructor***, sebaliknya constructor yang mempunyai lebih dari satu argumen adalah ***non-default consructor***. Constructor dengan satu default argument tetap merupakan sebuah default constructor,

```
class C
{
        public:
        C(int count=10) : _count(count) {}
        …
        private:
        int _count;
};
```

Compiler C++ dapat menambahkan ***default constructor*** bilamana diperlukan, jika dalam definisi class

- tidak tertulis secara eksplisit sebuah default constructor dan tidak ada deklarasi constructor lain (***copy constructor***).
- tidak ada anggota class berupa data const maupun ***reference***.

Sebagai contoh definisi class C sebagai berikut,

```
class C {…};
C c1; // memerlukan default constructor
```

```
C c2(c1); // memerlukan copy constructor
```

Compiler C++ memutuskan untuk menambahkan default dan copy construtor setelah menemui kedua baris program tersebut, sehingga definisi class secara efektif menjadi sebagai berikut,

```
class C
{
    public:
    C(); // default costructor
    C(const C& rhs); // copy constructor
    ~C(); // destructor
    C& operator=(const C& rhs); // assignment operator
    C* operator&(); // address-of operator
    const C* operator&(const C& rhs) const;
};
```

compiler menambahkan public constructor, dan destructor. Selain itu, compiler juga menambahkan ***assignment operator*** dan ***address-of operator***. Constructor (default dan non-default) tidak harus mempunyai akses public, sebagai contoh adalah pola desain (***design pattern***) Singleton.

```
class Singleton
{
    public:
    static Singleton* instance();
    protected:
    Singleton();
    private:
    static Singleton* _instance;
};
```

obyek (***instance***) ***singleton*** tidak dibentuk melalui constructor melainkan melalui fungsi instance. Tidak ada obyek ***singleton*** lain yang dapat dibentuk jika sudah ada satu obyek ***singleton***. Umumnya default constructor bentukan compiler (***generated default constructor***) menggunakan default constructor anggota bertipe class, sedangkan anggota biasa (***builtin type***) tidak diinisialisasi. Demikian halnya dengan obyek yang dibentuk dari obyek lain (copy), maka copy constructor bentukan compiler (***generated copy constructor***) menggunakan copy constructor dari anggota bertipe class pada saat inisialisasi. Sebagai contoh deklarasi class C berikut ini,

```
class C
{
        public:
        C(const char* aName);
        C(const string& aName);
        …
        private:
        std::string name;
};
```

copy constructor bentukan compiler menggunakan copy constructor class string untuk inisialisasi name dari aName. Jika class C tidak mempunyai constructor, maka compiler menambahkan juga default constructor untuk inisialisasi name menggunakan default constructor class string. Inisialisasi obyek menggunakan constructor (non-default) dapat dilakukan dengan ***member initializer*** maupun dengan ***assignment*** sebagai berikut,

member initialization

```
class C
{
        int i,j;
        public:
        C() : i(0),j(1) {}
        …
};
```

assignment

```
class C
{
        int i,j
        public:
        C()
        {
              i=0;j=0;
        }
        …
};
```

Kedua cara tersebut memberikan hasil yang sama, tidak ada perbedaan signifikan antara kedua cara tersebut untuk data bukan tipe class. Cara member initializer mutlak diperlukan untuk data const maupun ***reference***, seperti kedua contoh berikut ini:

```
class C //:1
{
	public:
	C(int hi,int lo) : _hi(hi),_lo(lo) {}
	…
	private:
	const int _hi,_lo; // const member
};
class C //:2
{
	public:
	C(const string& aName) : name(aName) {}
	…
	private:
	std::string& name; // reference member
};
```

Cara ***member initialization*** sebaiknya dilakukan untuk anggota bertipe class (***userdefined type***) seperti ditunjukkan pada contoh berikut ini,

```
class C
{
	public:
	C(const string& aName) : name(aName) { }
	private:
	std::string name; // bukan reference member
};
```

Pertimbangan menggunakan cara ***member initialization*** terletak pada efisiensi eksekusi program. Hal ini berkaitan dengan cara kerja C++ yang membentuk obyek dalam dua tahap,

- pertama, inisialisasi data
- kedua, eksekusi constructor (assignment)

Dengan demikian jika menggunakan cara assignment sebenarnya eksekusi program dilakukan dua kali, pertama inisialisasi kemudian assignment, sedangkan menggunakan ***member initialization*** hanya memanggil sekali constructor class string.

Semakin kompleks class tersebut (lebih kompleks dari class string) semakin mahal (tidak efisien) proses pembentukan obyek melalui cara assignment. Constructor dengan satu argumen berfungsi juga sebagai ***implicit conversion operator***. Sebagai contoh deklarasi class A dan B berikut ini,

```
class A
{
        public:
        A();
};
class B
{
        public:
        B(const A&);
};
```

pada cuplikan baris program di bawah ini terjadi konversi tipe obyek A ke B secara implisit melalui copy constructor class B.

```
A a
B b=a; // implicit conversion
```

explicit

C++ menyediakan satu sarana, menggunakan keyword explicit, untuk mengubah perilaku constructor dengan satu argumen agar tidak berfungsi sebagai ***conversion operator***. Jika class B menyatakan explicit pada copy constructor sebagai berikut,

```
class B
{
        public:
        explicit B(const A& a); // explicit ctor
};
```

maka konversi A ke B secara implisit tidak dapat dilakukan. Konversi A ke B dapat dilakukan secara eksplisit menggunakan ***typecast***,

```
A a;
B b=static_cast<B>(a); atau
B b=(B)a;
```

Konversi secara implisit dapat terjadi melalui argumen fungsi f dengan tipe B

```
void f(const B& );
```

tetapi f diakses dengan variabel tipe A, f(a). Apabila class B menghalangi konversi secara implisit maka argumen fungsi f menjadi,

```
f((B)a); atau
f(static_cast<B>(a));
```

Konversi tipe obyek secara implisit sebaiknya dihindari karena efeknya mungkin lebih besar terhadap aplikasi program secara keseluruhan dan tidak dapat dicegah pada saat kompilasi, karena construcor dengan argumen tunggal adalah suatu pernyataan program yang sah dan memang dibutuhkan.

Copy Constructor dan Copy Assignment

Sejauh ini sudah dibahas mengenai copy constructor sebagai anggota class yang berperan penting pada saat pembentukan obyek. Apabila sebuah class tidak menyatakan secara tegas copy constructor class tersebut, maka compiler menambahkan copy constructor dengan bentuk deklarasi,

```
C(const C& c);
```

Bentuk lain copy constructor adalah sebagai berikut,

```
C(C& c); atau
C(C volatile& c); atau
C(C const volatile& c);
```

Copy constructor class C adalah constructor yang mempunyai satu argumen. Sebuah copy constructor boleh mempunyai lebih dari satu argumen, asalkan argumen tersebut mempunyai nilai default (***default argument***).

```
C(C c); // bukan copy constructor
C(C const& c,A a=b); //copy constructor
```

Constructor dengan argumen bertipe C saja (tanpa ***reference***) bukan merupakan copy constructor. Copy constructor juga dibutuhkan pada saat memanggil suatu fungsi yang menerima argumen berupa obyek suatu class,

```
void f(C x);
```

memerlukan copy onstructor class C untuk mengcopy obyek c bertipe C ke obyek x dengan tipe yang sama, yaitu pada saat memanggil fungsi f(c)(***pass-by-value***).

Hal serupa terjadi pada saat fungsi f sebagai berikut,

```
C f()
{
```

```
C c;
…
return c;
}
```

mengirim obyek c ke fungsi lain yang memanggil fungsi f() tersebut. Copy assignment operator class C adalah operator=, sebuah fungsi yang mempunyai satu argumen bertipe C. Umumnya deklarasi copy assignment mempunyai bentuk,

```
C &operator=(const C &c);
```

Bentuk lain yang mungkin adalah,

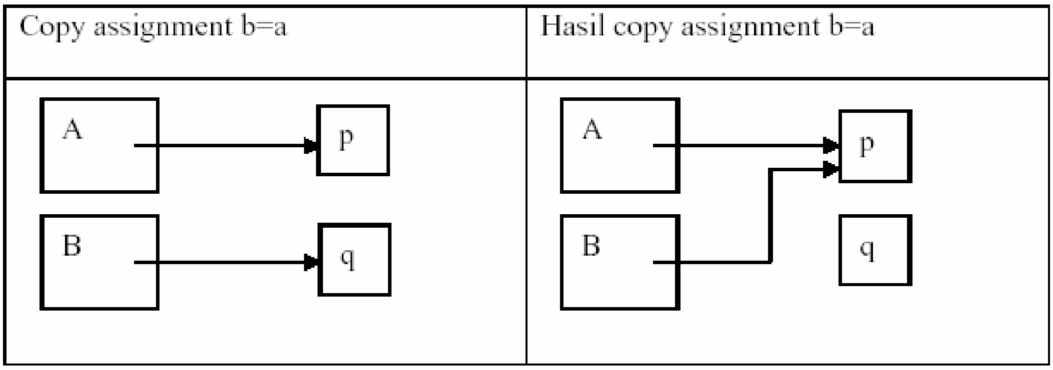
```
C &operator=(C &c); atau
C &operator=(C volatile &c); atau
C &operator=(C const volatile &c);
```

Copy assignment boleh mempunyai argumen dengan tipe C (bukan reference), tetapi tidak boleh mempunyai argumen lebih dari satu walaupun argumen tersebut mempunyai nilai ***default*** (default argument). Seperti halnya copy constructor, compiler akan menambahkan copy assignment jika suatu class tidak mempunyai fungsi tersebut. Copy assignment dibutuhkan untuk membentuk obyek melalui assignment, seperti contoh berikut:

```
class C
{
    public:
    C(); //ctor
    ~C(); //dtor
    …
};
C c1;
C c2=c1; //copy constructor
C c3;
c3=c1; //copy assignment
```

Class C tidak mempunyai copy constructor maupun copy assignment operator, maka pembentukan obyek c2, dan c3 menggunakan copy constructor dan copy assignment yang ditambahkan oleh compiler ke class C tersebut. Suatu class yang mempunyai data dengan alokasi dinamik (pointer) sebaiknya tidak mengandalkan copy constructor maupun copy assignment operator yang ditambahkan compiler. Copy assignment hasil tambahan compiler mengcopy (***memberwise copy***) pointer dari

obyek satu (yang dicopy) ke obyek lainnya (hasil copy), sehingga kedua obyek mengacu ke lokasi memori yang sama. Masalah timbul jika kedua obyek mempunyai masa pakai (***lifetime1***) yang berbeda. Jika salah satu obyek sudah habis masa pakainya maka destructor obyek tersebut mengembalikan memori (***dynamic memory***) yang digunakan obyek tersebut, padahal copy obyek tersebut masih mengacu ke lokasi memori yang sama.



Pada contoh hasil copy assignment b=a (***shalow copy***), menunjukkan kedua obyek a dan b mengacu ke lokasi memori p. Apabila obyek a melepas memori p (melalui destructor), maka obyek b mengacu ke lokasi memori yang sudah tidak valid lagi. Lokasi memori p dapat digunakan obyek lain jika obyek a melepasnya. Demikian pula halnya dengan lokasi memori q, apabila obyek b habis masa pakainya (keluar scope, dihapus dll) maka destructor class B tidak melepas memori q. Akibatnya terjadi pemborosan memori (***memory leak***). Salah satu jalan keluar adalah dengan menyatakan secara tegas copy constructor dan copy assignment yang dibutuhkan suatu class sehingga compiler tidak membuat copy constructor dan copy assignment ke class tersebut. Alternatif lain adalah menempatkan deklarasi copy constructor dan copy assignment operator private sebagai berikut,

```
class C
{ …
    private:
    C(const C&);
    C &operator=(const C&);
};
```

definisi copy constructor dan copy assignment operator class C pada contoh di atas tidak perlu ada, karena tujuannya adalah menghalangi proses penggandaan (copy) menggunakan kedua fungsi tersebut. Pada tahap kompilasi penggunaan assignment, b=a masih dapat diterima karena deklarasi asignment operator tersebut

tersedia. Pada saat link akan gagal karena ***linker*** tidak dapat menemukan definisi copy assignment operator. Teknik ini masih mempunyai kelemahan, karena class lain masih mungkin mempunyai akses ke private copy constructor dan copy assignment operator tersebut (melalui hubungan ***friendship***).

Destructor

Destructor adalah anggota class (***member function***) yang berfungsi melepas memori pada saat suatu obyek sudah tidak diperlukan lagi. Fungsi destructor kebalikan constructor. Destructor tidak mempunyai atau memerlukan argumen. Destructor juga tidak mengembalikan nilai apapun (tidak mempunyai ***return type***). Seperti halnya constructor, compiler dapat menambahkan sebuah destructor jika sebuah class tidak mempunyai destructor.

virtual Destructor

Sebuah destructor dapat berupa fungsi virtual. Hal ini menjadi keharusan jika class B,

- merupakan base class.
- class D yang menggunakan B sebagai base class mempunyai anggota berupa data dengan alokasi memori dinamik (pointer).

```
class B
{
      public:
      B();
      ~B();
};
class D : public B
{
      public:
      D() : p(new char[256]) {}
      ~D()
      {
          delete[] p;
      }
      …
      private:
      char *p;
};
```

Pada contoh tersebut destructor base class B bukan fungsi virtual. Dalam C++ umumnya obyek class D digunakan secara ***polimorphic*** dengan membentuk obyek

class D (***derived class***) dan menyimpan alamat obyek tersebut dalam pointer class B (***base class***) seperti pada contoh berikut ini,

```
void main(void)
{
        B *pB=new D();
        delete pB;
}
```

Dalam standar C++ menghapus obyek D (derived class) melalui pointer class B (base class) sedangkan destructor base class non-virtual mempunyai efek yang tidak menentu (***undefined behaviour***). Apabila standard C++ tidak menetapkan apa yang seharusnya berlaku, maka terserah kepada pembuat compiler menentukan perilaku program pada kondisi semacam ini. Umumnya pembuat compiler mengambil langkah untuk tidak memanggil destructor class D (derived class). Dengan demikian, pada saat menjalankan perintah delete, destructor class D tidak dieksekusi karena destructor base class B nonvirtual. Akibatnya lokasi memori dinamik yang digunakan class D tidak pernah dilepas. Hal ini adalah contoh lain terjadinya pemborosan memori (***memory leak***) oleh suatu program. Jalan keluarnya adalah membuat destructor base class B virtual,

```
class B
{
        public:
        B();
        virtual ~B();
}
```

Tidak seperti destructor, tidak ada ***virtual constructor*** atau ***virtual copy constructor***. Pada saat membentuk obyek, tipe obyek harus diketahui terlebih dahulu, apakah membentuk obyek class A, B, C dsb. Tidak ada aspek bahasa C++ untuk mewujudkan virtual constructor secara langsung, menempatkan virtual pada deklarasi constructor merupakan kesalahan yang terdeteksi pada proses kompilasi. Efek virtual constructor bukan tidak mungkin dicapai, C++ memungkinkan membuat idiom ***virtual constructor*** yang bertumpu pada fungsi virtual dalam kaitannya dengan hubungan antara sebuah class dengan ***base classnya***.

Ringkasan

Sejauh ini pembahasan artikel masih belum menyentuh aspek praktis pemrograman,

namun demikian dalam menterjemahkan suatu desain maupun memahami program yang ditulis orang lain sangatlah penting mengetahui aturan dasar sesuai standarisasi C++. Butir-butir pembahasan dalam artikel ini antara lain,

- Fokus pembahasan adalah aspek pembentukan obyek. Tidak membahas aturan (***rule***) berkaitan dengan class dalam C++ secara komprehensif.
- Constructor merupakan anggota class yang berperan dalam pembentukan obyek. Compiler menambahkan constructor bilamana diperlukan ke class yang tidak mempunyai constructor. Constructor tidak harus mempunyai akses public. Inisialisasi data menggunakan constructor dapat dilakukan dengan cara member initialization dan assignment. Keduanya tidak mempunyai perbedaan signifikan untuk data biasa (***built-in type*** seperti char, int, float, dll). Cara member initialization lebih efisien untuk data berupa class (***user-defined type***).
- Constructor dengan satu argumen dapat digunakan untuk konversi tipe data secara implisit. C++ menyediakan explicit untuk mengubah perilaku ini, karena hal tersebut melonggarkan janji C++ sebagai bahasa yang mengutamakan ***strict type (type safe)***.
- Sebuah class membutuhkan copy constructor dan copy assignment operator untuk menggandakan obyek suatu class. Hal ini terjadi juga pada saat memanggil suatu fungsi dengan cara ***pass-by-value***. Apabila suatu class tidak mempunyai copy constructor dan copy assignment maka compiler menambahkannya. Copy constructor dan copy assignment hasil tambahan compiler bekerja dengan cara ***memberwise copy*** dan menghasilkan ***shalow copy*** untuk data dengan alokasi memori dinamik.
- Destructor merupakan anggota class yang berfungsi pada saat ***lifetime*** suatu obyek habis. Destructor sebuah base class sebaiknya virtual.
- Constructor selalu merupakan fungsi non-virtual. Efek ***virtual constructor*** dan ***virtual copy constructor*** mungkin diperlukan dalam suatu desain. Efek virtual constructor dapat diwujudkan melalui sifat polimorphisme class. Efek virtual copy constructor dapat diwujudkan memanfaatkan aspek ***covariant return type*** sebuah hirarki class. Kedua hal tersebut memerlukan pembahasan khusus.
- Pembahasan pembentukan obyek belum dikaitkan dengan jenis scope yang ada dalam C++. C++ mempunyai jenis scope yang lebih kaya dibandingkan bahasa C, selain file scope, function scope, dan block scope C++ memiliki ***class scope*** dan namespace scope. Salah satu panduan praktis bahkan menyarankan untuk menunda (***lazy initialization***) pembentukan obyek selagi belum diperlukan.
- Pembentukan suatu obyek mungkin saja gagal. Artikel ini tidak membahas mengenai kegagalan pembentukan obyek, karena pembahasan tersebut berkaitan pembahasan ***exception*** dalam C++. Pembahasan exception C++ (***exception safety***) merupakan topik tersendiri.

Desain dan implementasi class C++ bukanlah hal yang mudah, masih banyak aspek lain yang belum terjangkau pembahasan artikel ini. Pada artikel selanjutnya akan dibahas scope (***visibility***) dalam C++, batasan akses (access specifier) C++, abstract class, function overloading, class relationship, template, dll.

13. Teknik Pencarian (Searching) dan Pengurutan (Sorting)

Penerapan dari konsep pemrograman sebelumnya, dapat digunakan untuk berbagai macam permasalahan. Di bab ini akan dibahas penerapan ke dalam teknik pencarian (***searching***) dan pengurutan (***sorting***).

PENCARIAN (SEARCHING)

Pencarian (***Searching***) merupakan proses yang fundamental dalam pemrograman, guna menemukan data (nilai) tertentu di dalam sekumpulan data yang bertipe sama. Fungsi pencarian itu sendiri adalah untuk memvalidasi (mencocokkan) data. Sebagai contoh, untuk menghapus atau mengubah sebuah data di dalam sekumpulan nilai, langkah pertama yang harus di tempuh adalah mencari data tersebut, lalu menghapus atau mengubahnya. Contoh lain adalah penyisipan data ke dalam kumpulan data, jika data telah ada, maka data tersebut tidak akan disisipkan, selainnya akan disisipkan ke kumpulan data tersebut.

Ada sebuah kasus sederhana, misalkan terdapat 10 data yang bertipe integer, terangkum di dalam variabel larik L. Terdapat data X di dalam larik L tersebut. Bagaimana proses pencarian data X tersebut?. Jika ketemu maka akan mengeluarkan pesan teks “Data ditemukan!” atau jika tidak ditemukan akan mengeluarkan pesan teks “Data tidak ditemukan“. Serta menampilkan di elemen ke berapa elemen tersebut ditemukan, dan berapa jumlah data X di larik L.

Ada beberapa metode mencari data di dalam sekumpulan data yang bertipe sama, yaitu:

1. Metode Pencarian Beruntun (***Sequential Search***)
2. Metode Pencarian Bagi dua (***Binary Search***)

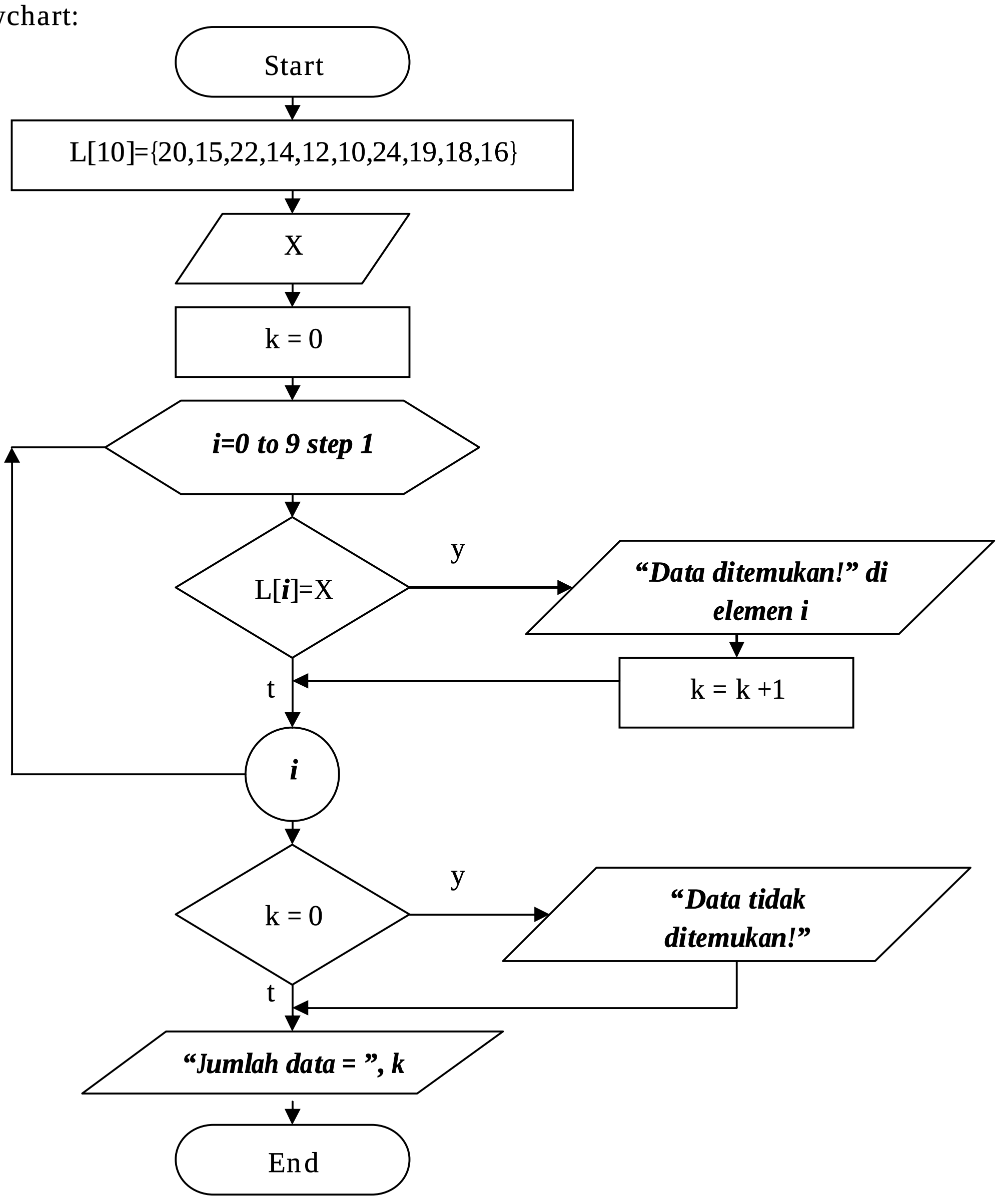
Metode Pencarian Beruntun

Konsep yang digunakan dalam metode ini adalah membandingkan data-data yang ada dalam kumpulan tersebut, mulai dari elemen pertama sampai elemen ditemukan, atau sampai elemen terakhir. Perhatikan alur di bawah ini:
Misalkan kumpulan data L tersebut adalah sebagai berikut:

20	15	22	14	12	10	24	19	18	16

Data yang akan dicari adalah X, misal X = 10, maka elemen yang diperiksa adalah elemen yang bernilai 10.

Flowchart:



Keterangan:

Dari flowchart di atas dapat dilihat bahwa pada saat terjadi perulangan, nilai ***L[i]*** akan diperiksa, apakah sama dengan nilai ***X*** yang diinput. Jika nilainya sama, maka akan mengeluarkan pesan ***"Data ditemukan di elemen i"*** dan langsung menambahkan variabel ***k*** dengan nilai 1. Apa fungsi variabel ***k*** disini?. Variabel ***k*** berfungsi untuk menjumlah ada berapa banyak nilai X yang terdapat di larik L, karena itulah ada nilai awal, yaitu ***k = 0.*** Setelah perulangan selesai, maka akan diperiksa apakah jumlah ***k = 0***? Jika bernilai benar, maka akan mengeluarkan pesan "Data tidak ditemukan". Dan sebelum flowchar berakhir, program akan mengeleuarkan jumlah nilai X di larik L.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
	int X,i,k;
	int L[10] = {20,15,22,14,12,10,24,19,18,16};

	printf("Data yang akan dicari = ");scanf("%d",&X);
	k = 0;
	for(i=0;i<=9;i++)
	{
		if(L[i]==X)
		{
			printf("Data ditemukan di elemen %d \n",i);
			k++;
		}
	}
	if(k==0)
	{
```

```
        printf("Data tidak ditemukan \n");
    }
    printf("Jumlah data yang ditemukan = %d",k);

    getch();
    return 0;
}
//--------------------------------------------------------------------------
```

Secara umum, pencarian beruntun kinerjanya relatif lambat, dikarenakan adanya proses perulangan dalam metode tersebut. Bayangkan jika ada lebih dari 100.000 data, artinya akan ada 100.000 kali perulangan. Jika dalam proses satu perulangan membutuhkan 0,01 detik, maka akan memakan waktu sekitar 1000 detik atau 16,7 menit. Karena itulah, untuk pencarian dengan data yang besar, metode ini tidak digunakan oleh progammer.

Metode Pencarian Bagi Dua (Binary Search)

Metode ini diterapkan pada sekumpulan data yang sudah terurut (menaik atau menurun). Metode ini lebih cepat dibandingkan metode pencarian beruntun. Data yang sudah terurut menjadi syarat mutlak untuk menggunakan metode ini.

Konsep dasar metode ini adalah membagi 2 jumlah elemennya, dan menentukan apakah data yang berada pada elemen paling tengah bernilai sama, lebih dari atau kurang dari nilai data yang akan dicari. Jika bernilai sama, maka langsung data yang dicari ditemukan. Jika data di elemen terurut naik, maka jika data yang berada di tengah kurang dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kanan, dan begitu seterusnya sampai ketemu atau tidak sama sekali. Dan sebaliknya untuk nilai data yang berada di tengah lebih dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kiri, dan begitu seterusnya sampai ketemu atau tidak sama sekali. Dan demikian sebaliknya untuk data yang terurut menurun. Dalam hal ini tentukan indeks paling awal dan indeks paling akhir, untuk membagi 2 elemen tersebut.

Indeks ***awal = i***, dimana nilai i, pada awalnya bernilai 0;
Indeks ***akhir = j***, dimana nilai j, pada awalnya bernilai sama dengan jumlah elemen.

Langkah-langkah untuk metode pencarian bagi dua.

1. Asumsikan data terurut secara horizontal dari indeks 0 sampai ***n-1***, untuk menggunakan istilah kanan dan kiri.
2. Misalkan kumpulan data yang berjumlah ***n*** adalah larik ***L***, dan data yang akan dicari adalah ***X***.
3. Tentukan nilai indeks awal ***i = 0*** dan indeks akhir ***j = n-1***.
4. Tentukan apakah data terurut menurun atau menaik dengan menggunakan membandingkan apakah elemen paling kiri ***L[0]*** lebih dari atau kurang dari elemen paling kanan ***L[n-1]***.
   - Jika data di elemen paling kiri ***L[0]*** > data di elemen paling kanan ***L[n-1],*** maka data terurut menurun.
   - Jika data di elemen paling kiri ***L[0]*** < data di elemen paling kanan ***L[n-1],*** maka data terurut menaik.
5. Asumsikan bahwa data terurut menaik (tergantung hasil dari nomor 3).
6. Misalkan variabel k adalah indeks paling tengah, diperoleh dengan rumus:

   k = (i + j) div 2.

7. Periksa, jika ***L[k] = X,*** maka data dicari langsung ketemu di elemen ***k***.
8. Jika nomor 7 tidak terpenuhi, periksa jika ***L[k] < X***, maka pencarian berikutnya dilakukan di sisi kanan indeks k, lakukan proses seperti pada nomor 6, dimana nilai indeks ***i*** sekarang sama dengan nilai indeks ***k*** sebelumnya.

   i = k.
   k = (i + j) div 2.

   Dan seterusnya sampai nilai X dicari ketemu atau tidak sama sekali.

9. Jika nomor 8 tidak terpenuhi, maka tidak pasti nilai ***L[k] > X***, maka pencarian berikutnya dilakukan di sisi kiri indeks k, lakukan proses seperti pada nomor 6, dimana nilai indeks ***j*** sekarang sama dengan nilai indeks ***k*** sebelumnya.

   j = k.
   k = (i + j) div 2.

   Dan seterusnya sampai nilai X dicari ketemu atau tidak sama sekali.

10. Jika data terurut menurun, maka tukar kondisi yang ada di nomor 8 dan 9.

Contoh:
Diberikan 10 data terurut L[10] = {12,14,15,17,23,25,45,67,68,70}. Cari nilai ***X = 14*** di elemen tersebut.

Solusi:

1. Menentukan apakah data terurut menaik atau menurun.
   L[0] = 12
   L[9] = 70
   Karena ***L[0] < L[9],*** maka data tersebut terurut menaik.

2. Misal indeks paling kiri adalah i = 0 dan indeks paling kanan adalah j = 9, maka indeks tengahnya adalah :

   k = (i + j) div 2
   =(0 + 9) div 2
   = 4.

   Elemen tengah sekarang adalah 4 dengan L[4] = 23.

3. Karena data di indeks tengah lebih dari nilai data yang dicari (***L[4] > X***), maka pencarian berikutnya dilakukan pada sisi kiri indeks k, maka nilai j sekarang sama dengan k, lalu lakukan proses sama seperti nomor 2.

   j = k
   = 4
   k = (i + j) div 2
   = (0 + 4) div 2
   = 2.

   Elemen tengah sekarang adalah 2 dengan L[2] = 15.

4. Karena data di indeks tengah lebih dari nilai data yang dicari (***L[2] > X***), maka pencarian berikutnya dilakukan pada sisi kiri indeks k, maka nilai j sekarang sama dengan k, lalu lakukan proses sama seperti nomor 2.

   j = k
   = 2
   k = (i + j) div 2
   = (0 + 2) div 2
   = 1.
   Elemen tengah sekarang adalah 1 dengan L[1] = 14.

5. Karena nilai data di elemen tengah sama dengan nilai data yang dicari X, maka pencarian berakhir. Data X ditemukan di indeks ke-1.

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
	int X,i,j,k,p;
	int L[10] = {12,14,15,17,23,25,45,67,68,70};

	/* Menentukan apakah terurut menaik atau menurun */
	/* variabel p digunakan untuk kode, apakah menaik atau menurun */
	/* jika p = 0, maka data terurut menaik */
	/* jika p = 1, maka data terurut menurun */

	if (L[0]<L[9])
	{
		printf("Data terurut menaik \n");
		p = 0;
	}
	else
	{
		printf("Data terurut menurun \n");
		p = 1;
	}

	/* input data X yang akan dicari */

	printf("Data yang akan dicari = ");scanf("%d",&X);

	/* penentuan indeks awal dan akhir semula */
```

```
i = 0;
j = 9;

/* proses perulangan untuk menentukan nilai tengah k */

do
{
    k = (i + j) / 2;
    if (p==0)     // jika data terurut menaik
    {
        if (L[k]==X)
        {
            printf("Data ditemukan di elemen %d",k);

            getch();
            return 0;     // langsung keluar program
        }
        else if (L[k]< X)
        {
            i = k;
        }
        else
        {
            j = k;
        }
    }
    else      // jika data terurut menurun
    {
        if (L[k]==X)
        {
            printf("Data ditemukan di elemen %d",k);

            getch();
            return 0;     // langsung keluar program
        }
        else if (L[k]> X)
        {
            i = k;
```

```
            }
            else
            {
                j = k;
            }
        }
    }
    while(k!=0);    // sampai nilai k = 0, iterasi berakhir

    printf("Data tidak ditemukan!");

    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

Terlihat dari contoh kasus di atas, bahwa perulangan terjadi tidak sebanyak di metode pencarian beruntun, karena perulangan hanya dilakukan apabila data tidak ketemu, pada pembagian indeks elemen. Artinya pada metode pencarian beruntun, perulangan dilakukan sebanyak jumlah data, sedangkann di metode bagi dua, perulangan dilakukan sebanyak $^{2}\log(n)$ kali, sehingga metode pencarian bagi dua lebih cepat dibandingkan dengan metode pencarian beruntun.

PENGURUTAN (SORTING)

Pengurutan (sorting) adalah proses mengatur sekumpulan obyek menurut urutan atau susunan tertentu [WIR76]. Urutan tersebut dapat menaik (***ascending***) atau menurun (***descending***). Jika diberikan n buah elemen disimpan di dalam larik L, maka:

- pengurutan menaik adalah $L[0] \le L[1] \le L[2] \le ... \le L[n-1]$
- pengurutan menurun adalah $L[0] \ge L[1] \ge L[2] \ge ... \ge L[n-1]$.

Pengurutan berdasarkan jenisnya, dibagi dua kategori, yaitu:

1. Pengurutan Internal, yaitu pengurutan terhadap sekumpulan data yang disimpan di dalam memori utama komputer.
2. Pengurutan Eksternal, yaitu pengurutan data yang disimpan di dalam memori sekunder, biasanya data bervolume besar sehingga tidak mampu dimuat

semuanya dalam memori komputer, disebut juga pengurutan arsip (***file***), karena struktur eksternal yang dipakai adalah arsip.

Karena pengaksesan memori utama lebih cepat daripadamemori skunder, maka pengurutan internal lebih cepat daripada pengurutan eksternal.

Bermacam-macam metode yang dipakai untuk melakukan pengurutan, antara lain:

- Bubble Sort
- Selection Sort (Maximum & Minimum Sort)
- Insertion Sort
- Heap Sort
- Shell Sort
- Quick Sort
- Merge Sort
- Radix Sort
- Tree Sort.

Di bab ini hanya akan dibahas mengenai tiga buah metode sederhana yang mendasar, yaitu:

1. Metode Pengurutan Gelembung (***Bubble Sort***)
2. Metode Pengurutan Piih (***Selection Sort***)
3. Metode Pengurutan Sisip (***Insertion Sort***).

Metode Pengurutan Gelembung (***Bubble Sort***)

Metode ini diinspirasi oleh gelembung sabun yang berada di permukaan air. Karena berat jenis gelembung sabun lebih ringan dibandingkan dengan berat jenis air, sehingga gelembung sabun selalu terapung di permukaan air. Prinsip pengapungan inilah yang diterapkan ke metode ini, dimana nilai yang paling rendah berada di posisi paling atas, melalui proses pertukaran.

Konsep dasar dari metode ini adalah setiap data yang ada di kumpulan, dibandingkan dengan data-data lainnya, artinya jika jumlah data sebanyak 5, maka akan terjadi pembandingan sebanyak ***$(5-1)^2$ = 16 kali***. Untuk satu data, akan dibandingakan sebanyak 4 kali terhadap data yang lainnya.

Atau secara umum dapat ditarik rumus, untuk jumlah data sebanyak n buah, maka:

$$jumlahiterasi\ pembandingan = (n-1)^2$$

Jika data-data tersebut disimpan di dalam lari L, maka:

1. untuk pengurutan menaik, pembandingannya sebagai berikut:

$$L[n] < L[n-1]$$

2. untuk pengurutan menurun, pembandingannya sebagai berikut:

$$L[n] > L[n-1].$$

Jika kondisi di atas terpenuhi, maka nilai data yang ada di indeks ***n-1*** akan ditukar dengan nilai data yang ada di indeks ***n***.

Perhatikan program pengurutan menaik di bawah ini:

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
    int i,j,k;
    int temp;
    int L[10] = {20,15,22,14,12,10,24,19,18,16};

    for(i=1;i<=9;i++)
    {
        for(j=9;j>=1;j--)
        {
            if(L[j]<L[j-1])
            {
                temp=L[j];
                L[j]=L[j-1];
                L[j-1]=temp;
            }
```

```
        }
    }

    for(i=0;i<=9;i++)
    {
        printf("%d ",L[i]);
    }

    getch();
    return 0;
}
//-----------------------------------------------------------------------
```

Terlihat dari program di atas, bahwa terjadi perulangan di dalam perulangan (***nested looping***). Hal ini menunjukkan bahwa ada proses pembandingan sebanyak ***(10-1)² = 81 kali***, dimana setiap data dibandingkan dengan 8 data lainnya. Jika kondisi perbandingan terpenuhi, maka akan menukarkan posisi (indeks) data di larik L. Karena perbandingan menggunakan operator pembanding kurang dari (<), maka akan menghasilkan data larik ***L*** yang terurut menaik.

Metode ini relatif lebih lambat, karena banyaknya terjadi iterasi perbandingan yang dilakukannya, namun lebih mudah dipahami.

Latihan : Buat program untuk menghasilkan data yang terurut menurun (dari kasus di atas).

Metode Pengurutan Pilih (***Selection Sort***)

Metode ini memiliki konsep memilih data yang maksimum/minimum dari suatu kumpulan data larik L, lalu menempatkan data tersebut ke elemen paling akhir atau paling awal sesuai pengurutan yang diinginkan. Data maksimum/minimum yang diperoleh, diasingkan ke tempat lain, dan tidak diikutsertakan pada proses pencarian data maksimum/minimum berikutnya. Perhatikan ilustrasi berikut:

Misalkan ada sekumpulan data acak berjumlah ***n*** elemen yang disimpan di dalam larik ***L,*** akan diurut menaik, maka langkah-langkah yang harus dilakukan adalah:

1. Menentukan jumlah iterasi, yaitu ***pass = n – 2.***
2. Untuk setiap pass ***ke-i = 0,1,2,...,pass***, lakukan:

a. Cari elemen terbesar (***maks***) dari elemen ***ke-i*** sampai ***ke-(n-1).***
b. Pertukarkan maks dengan elemen ke-i.
c. Kurangin ***n*** satu ***(n = n – 1).***

Rincian tiap-tiap pas adalah sebagai berikut:

- pass 0
  - Cari elemen maksimum di dalam ***L[0...(n-1)].***
  - Pertukarakan elemen maksimum dengan elemen ***L[n-1].***

- pass 1
  - Cari elemen maksimum di dalam ***L[0...(n-2)].***
  - Pertukarakan elemen maksimum dengan elemen ***L[n-2].***

- pass 2
  - Cari elemen maksimum di dalam ***L[0...(n-3)].***
  - Pertukarakan elemen maksimum dengan elemen ***L[n-3].***

.
.
.

- pass 3
  - Cari elemen maksimum di dalam ***L[0...1].***
  - Pertukarakan elemen maksimum dengan elemen ***L[1].***

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
      int i,j;
      int Imaks, maks, temp;
      int L[10] = {20,15,22,14,12,10,24,19,18,16};

      for(i=9;i>=1;i--)
```

```
	{
		Imaks = 0;
		maks = L[0];
		for(j=1;j<=i;j++)
		{
			if(L[j]>maks)
			{
				Imaks = j;
				maks = L[j];
			}
		}
		temp=L[i];
		L[i]=maks;
		L[Imaks]=temp;
	}

	for(i=0;i<=9;i++)
	{
		printf("%d ",L[i]);
	}

	getch();
	return 0;
}
//--------------------------------------------------------------------------
```

Seara garis besar, metode ini relatif lebih cepat dibandingkan dengan metode gelembung, karena operasi pertukaran hanya sekali dilakukan pada setiap pass, sehingga waktu pengurutan dapat dikurangi.

Latihan : Buat program untuk pengurutan menurun (contoh kasus sama dengan di atas).

Metode Pengurutan Sisip (***Insertion Sort***)

Metode ini dilakukan dengan cara menyisipkan elemen larik pada posisi yang tepta. Pencarian posisi yang tepat dilakukan dengan melakukan pencarian beruntun di dalam larik.

Perhatikan tahap-tahap di bawah ini:

Misalkan ada sekumpulan data acak berjumlah ***n*** elemen yang disimpan di dalam larik ***L,*** akan diurut menaik, maka langkah-langkah yang harus dilakukan adalah:

1. Untuk setiap pass ***ke-i = 1,2,...,n-1*** lakukan:
    a. X = L[i]
    b. Sisipkan ***x*** pada tempat yang sesuai anatara ***L[0]...L[i].***

    Rincian tiap-tiap pass adalah sebagai berikut:

    Dianggap pass 0 : L[0] dianggap sudah pada tempatnya.

    - pass 1
      x = L[1] harus dicari tempatnya yang tepat pada ***L[0..1]*** dengan cara menggeser elemen ***L[0..0]*** ke kanan bila ***L[0..0]*** lebih besar daripada ***L[1].*** Misalkan posisi yang tepat adalah ***k***, sisipkan ***L[1]*** pada ***L[k].***

    - pass 2
      x = L[2] harus dicari tempatnya yang tepat pada ***L[0..2]*** dengan cara menggeser elemen ***L[0..1]*** ke kanan bila ***L[0..1]*** lebih besar daripada ***L[2].*** Misalkan posisi yang tepat adalah ***k***, sisipkan ***L[2]*** pada ***L[k].***

    .
    .
    - pass n-1
      x = L[n-1] harus dicari tempatnya yang tepat pada ***L[0..(n-1)]*** dengan cara menggeser elemen ***L[0..(n-2)]*** ke kanan bila ***L[0..(n-2)]*** lebih besar daripada ***L[n-1].*** Misalkan posisi yang tepat adalah ***k***, sisipkan ***L[n-1]*** pada ***L[k].***

Program:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <conio.h>
```

```
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
	int i,j;
	int x;	// variabel bantu agar L[k] tidak ditimpa (replace) selama pergeseran
	bool ketemu;

	int L[10] = {20,15,22,14,12,10,24,19,18,16};

	for(i=1;i<=9;i++)
	{
		/* sisipkan L[i] ke dalam bagian yang sudah terurut */
		x = L[i];

		/*cari posisi yang tepat untuk x di dalam L[0..i-1] sambil menggeser */
		j=i-1;
		ketemu = false;

		while ((j>=0)&&(!ketemu))
		{
			if(x < L[j])
			{
				L[j+1] = L[j];
				j--;
			}
			else
			{
				ketemu = true;
			}
		}
		/* j < 1 or ketemu */
		L[j+1] = x;
	}

	for(i=0;i<=9;i++)
	{
		printf("%d ",L[i]);
	}
```

```
    getch();
    return 0;
}
//---------------------------------------------------------------------------
```

Kelemahan metode ini terletak pada banyaknya operasi pergeseran yang diperlukan dalam mencari posisi yang tepat untuk elemen larik. Pada setiap pass ke-1, operasi pergeseran yang diperlukan maksimum ***i-1 kali***. Untuk larik dengan n yang besar, jumlah operasi pergeseran meningkat secara kuadratik, sehingga pengurutan sisip tidak praktis untuk volume data yang besar.

Latihan: Buat program pengurutan menurun untuk kasus yang sama seperti program di atas.

14. Tips dan Triks

Dari berbagai pengalaman, ada hal-hal yang penting untuk mempelajari konsep dan aplikasi pemrograman serta dengan menggunakan software Borland C++ Builder 6 ini. Disini akan dituangkan dalam bentuk Tips dan Triks, karena mungkin yang membaca dan mempelajari buku ini, akan menemukan hal-hal yang baru yang lebih efektif dan kreatif.

1. Sebelum memulai setiap tugas dan pekerjaan, luruskan niat dan tujuan serta berdoalah memohon pertolongan untuk dibukakan kemudahan dalam melaksanakan tugas dan pekerjaan tersebut.

2. Pahami dulu konsep pemrograman, sebelum memulai mempelajari bahasa pemrogramannya. Pelajari konsep dari contoh-contoh yang terjadi di sekitar lingkungan sendiri yang paling dekat dan mudah dimengerti. Dan akan sangat lebih mudah jika sudah dekat dengan dasar matematika, terutama Matematika Diskret.

3. Setelah konsep pemrograman dimengerti, pilihlah bahasa pemrograman yang cocok dan bersifat universal, artinya dari segi pemrograman banyak digunakan oleh software-software lain.

4. Perbanyak jam terbang dan pengalaman di bahasa pemrograman tersebut, carilah contoh-contoh kasus yang dapat membantu dalam proses pemahaman bahasa pemrograman tersebut.

5. Setelah memahami satu kasus, berbagilah pengalaman dan ilmu ke sahabat paling dekat, karena dari merekalah terdapat masukkan berupa kritik dan saran yang akan menambah lagi pemahaman tentang itu.

6. Jangan menganggap enteng dan mudah setiap kasus, karena akan membutakan pengetahuan yang telah didapat selama ini, dan jangan kecewa jika pekerjaan tidak dihargai orang lain, ketahuilah karena orang yang tidak menghargai pekerjaan orang lain sebenarnya tidak mempunyai kemampuan dan pengetuahuan sedikitpun mengenai pekerjaan itu.

7. Setelah setiap menyelesaikan tugas dan pekerjaan, berdoalah dan bersyukurlah untuk keberkahan terhadap ilmu dan pengetahuan yang telah dipelahari.

15. Penutup

Konsep pemrograman merupakan pondasi awal untuk mempelajari suatu bahasa pemrograman. Setiap kasus yang ada di sekitar lingkungan sekitar, perlu pemahaman terhadap permasalahan dari kasus tersebut. Dalam hal pemahaman konsep pemrograman, berbagai metode dapat ditempuh dan dijalani, misalkan dengan studi kasus, diagram alir dan survei data. Pada intinya, metode apapun yang digunakan untuk memahami mengenai pemrograman, dibutuhkan ketekunan dan pengalaman yang lebih, karena setiap kali menemukan sebuah kasus, maka akan bertambah pengalaman yang akan didapat.

Isi materi yang terdapat di dalam buku ini hanya meruapakan dasar dan sebagian kecil yang terdapat di softwatre Borland C++ Builder 6. Masih banyak fitur-fitur yang perlu diketahui. Setelah buku ini terbit, maka akan dibuat buku yang membahas mengenai aplikasi visual (***visual applicataion***), khususnya mengenai aplikasi yang berorientasi database (***database oriented programming***) dan konsep perancangan software sistem informasi.

Untuk selanjutnya, proses pengembangan terhadap pemrograman khususnya menggunakan Borland C++ Builder ini, bergantung pada kreatifitas masing-masing. Bahasa C ini, seperti telah dikemukakan di muqaddimah, sangat memiliki relasi dan keterkaitan dengan software lain, misalnya Sistem Operasi Linux, Webserver Apache, Web Programming PHP, Java, Macromedia Flash, dsb. Wallahu A'lam.

Referensi

Al-Qur'an & Al-Hadits.

Heriyanto, Imam, Budi Raharjo (2003). ***Pemrograman Borland C++ Builder***. Informatika Bandung..

Indrajit, Richardus Eko. ***Manajemen Sistem Informasi dan Teknologi Informasi***.

Indrajit, Richardus Eko. ***Kajian Strategis Analisa Cost-Benefit Investasi Teknologi Informasi***.

Lidya, Leoni, rinaldi Munir (2002). ***Algoritama dan Pemrograman dalam Bahas Pascal dan C***. Informatika Bandung.

Solichin, Achmad (2003). ***Pemrograman Bahasa C dengan Turbo C***. IlmuKomputer.Com.

Wahono, Romi Satria(2003). ***Cepat MahirBahasa***. IlmuKomputer.Com.

Riwayat Hidup



Muhammad Fachrurrozi dilahirkan di Palembang, 22 Mei 1980. Saat ini mengabdikan dirinya untuk sebuah lembaga pendidikan yaitu Program Diploma Komputer Universitas Sriwijaya di Palembang sebagai Tenaga Laboran dan Staff Pengajar. Menyelesaikan studi sarjananya di Jurusan Matematika Fakultas MIPA Universitas Sriwijaya Desember 2002. Memulai debutnya di bidang komputer pada usia 11 tahun, dengan berbagai pengalaman selama mendalami dan memahani komputer baik dari sisi hardware maupun software. Saat duduk di bangku kuliah semester 3, ia telah dipercaya untuk menjadi salah satu asisten laboratorium komputer di jurusannya di bidang pemrograman. Saat waktu di bangku kuliah juga ia telah ditempa berbagai pengalaman dalam bidang teknologi informasi, yaitu pada tahun 2001 dan 2002 dipercaya sebagai asissten dan instruktur untuk Pelatihan Internet dan Database untuk pegawai Pendidikan Nasional se-Sumsel. Sejak setelah menamatkan kuliahnya ia mengabdikan diri di Program Diploma Komputer Universitas Sriwijaya. Sehari-hari dapat dihubungi melalui email ***m_fach@unsri.ac.id*** .